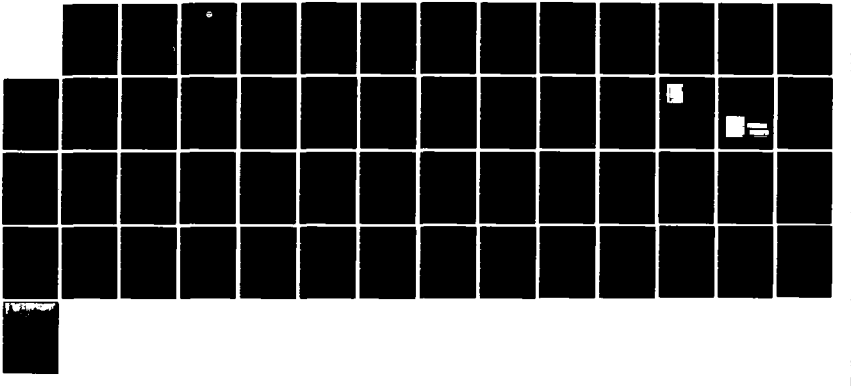
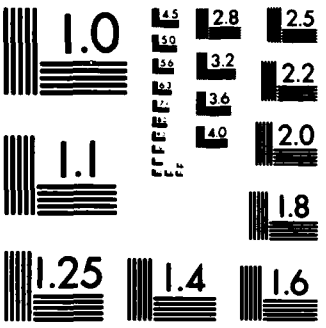
AD-A130 590 NOSC SYSTOLIC PROCESSOR TESTBED(U) NAVAL OCEAN SYSTEMS CENTER SAN DIEG CA J J SYMANSKI 0 JUN 83 NOSC/TD588 1/1

UNCLASSIFIED F/G 9/ NL





NOSC TD 588

NOSC TD 588

Technical Document 588

NOSC SYSTOLIC PROCESSOR TESTBED

J. J. Symanski

1 June 1983

Approved for public release; distribution unlimited

NOSC

NAVAL OCEAN SYSTEMS CENTER
San Diego, California 92152

NAVAL OCEAN SYSTEMS CENTER, SAN DIEGO, CA 92152

AN ACTIVITY OF THE NAVAL MATERIAL COMMAND

JM PATTON, CAPT, USN
Commander

HL BLOOD
Technical Director

ADMINISTRATIVE INFORMATION

The work covered in this report was done during fiscal years 1981 and 1982 under the NOSC Independent Research and Independent Exploratory Development program. The Naval Electronic Systems Command is sponsoring the follow-on algorithm and architectural development utilizing the systolic testbed.

Released by
M.S. Kvigne, Head
Communications Research
and Technology Division

Under authority of
H.D. Smith, Head
Communications Systems
and Technology Department

Since this is an on-going development, the information in this document is constantly being revised and improved. Any prospective users should contact the author if updated information is desired.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER NOSC Technical Document 588 (TD 588)	2. GOVT ACCESSION NO. AD-A130590	3. RECIPIENT'S CATALOG NUMBER
4. TITLE *(and Subtitle)* NOSC Systolic Processor Testbed		5. TYPE OF REPORT & PERIOD COVERED Fiscal years 1981–1982
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) J.J. Symanski		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Ocean Systems Center San Diego, CA 92152		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS		12. REPORT DATE 1 June 1983
		13. NUMBER OF PAGES 57
14. MONITORING AGENCY NAME & ADDRESS*(if different from Controlling Office)*		15. SECURITY CLASS. *(of this report)* Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT *(of this Report)*

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

18. SUPPLEMENTARY NOTES

19 KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

Systolic array
Testbed design
High speed signal processing

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

NOSC has expended considerable resources in designing and fabricating a systolic array testbed and writing software utility programs for its exploitation. These tools now permit us to readily explore the hardware design tradeoffs for future generation systolic processors. However, we do not have the resources to explore all desirable paths. It will be to the Navy's benefit if the tools we do have are made available to other U.S. researchers (probably via telephone/modem access) for them to explore their particular algorithmic interests. This document describes the capabilities and usefulness of these tools.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

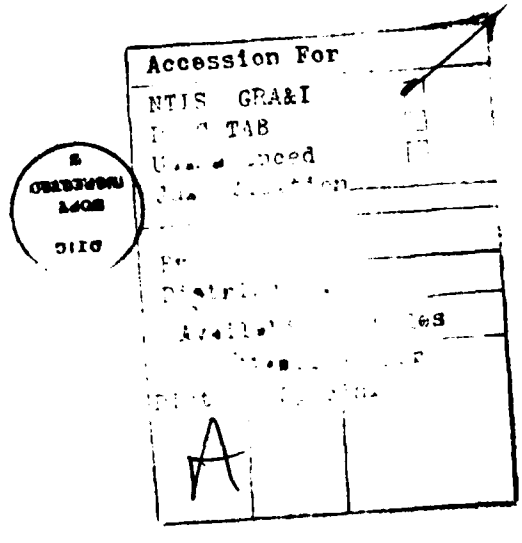
S N 0102-LF-014-6601

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

CONTENTS

APPENDICES



INTRODUCTION

In November 1978, Professor H. T. Kung of Carnegie-Mellon University presented a paper called "Systolic Arrays for VLSI" at the Symposium on Sparse Matrix Computations and Their Applications. Only with the reprinting of this paper in Introduction to VLSI Systems by Mead and Conway (Addison Wesley Publishing Company, 1980) has the importance of this concept to high-speed signal processing and VLSI integrated circuit designs been recognized.

Systolic architectures appear very attractive for the efficient computation of a large variety of matrix-intensive calculations for Navy high-speed signal processing needs. The systolic-array concept involves the inherent high throughput and simplicity offered by a lattice of identical processing elements, all operating in parallel on data flowing through the structure. The prospects for fabricating an array element (or several elements) on a single chip appear very good. However, many details of the algorithms, data flow, control, input/ output (I/O) schemes, numerical accuracy, speed, etc. have to be determined before a particular chip design can be undertaken.

The goal of this work is to build a systolic array testbed which is flexible enough to allow experimentation with algorithms and configurations so that intelligent decisions can be made when it comes time to specify the chip architecture for a particular set of applications. The testbed was not designed for optimum parameters for a specific application, but for flexibility in evaluation of various algorithms and architectures which may be implemented in the future. The testbed has input/output limitations due to the single I/O channel from the host system (a common problem for array processors) as well as the speed limitation set by the relatively slow arithmetic processing unit. These limitations will not be present in future implementations using custom VLSI devices.

HARDWARE DESCRIPTION

Since the systolic array concept was just beginning to gain momentum in 1979 when we began the testbed design, we decided to design for maximum flexibility to give us the capability of studying a wide variety of algorithms and architectures. The two biggest factors which allow this flexibility are (1) the use of a microprocessor within the systolic processing element (SPE)

and (2) the use of a minicomputer, with the power of a high level language, to control the array of processors.

The NOSC systolic array testbed uses a Hewlett-Packard 1000 minicomputer system as a host to control the systolic array. The HP 1000 system is composed of the usual complement of terminals, disc storage, hard copy printer, and other peripherals. The systolic array consists of an array of 64 processing elements interfaced to the HP 1000 through an interface which interprets commands from the host, stores data for the array and controls the transmission of data back to the host. More details of the testbed hardware are given in the appendices.

SOFTWARE DESCRIPTION

Several programs have been developed as tools to aid the operator in using the systolic array. The main software tools are (1) the HP editor, (2) the systolic processor compiler (SPC), (3) the systolic processor data entry (SPD), (4) the systolic processor file manager (SPF), and (5) the systolic processor execution program (SPX).

HP Editor

The HP 1000 editor (EDIT/1000) is used to enter instructions in the same manner as FORTRAN source code would be entered. This was done to utilize familiarity with the editor and to allow easy documentation of the testbed programs.

Systolic Processor Data (SPD)

Data for the array can be obtained in several ways: (1) operator entry, (2) loading from magnetic tape, or (3) by generation within the host itself. Data entered by any of the above means must be formatted by the host into a form which can be transferred to the testbed in an efficient manner.

Systolic Processor File Manager (SPF)

Over a period of time, an operator will create several data and program files. A special file management program was written to keep track of these files. The SPF program allows the operator to get a list of all the data or

source code files he has generated on his private disc or that are available on the group disc from other users. Listings can be obtained of the names only or of names and the first line of the comments stored with each file.

Systolic Processor Compiler Program (SPC)

Once the operator has entered his program using HP EDIT/1000, the SPC program is used to convert the text file into a binary file which can be sent to the systolic array processor (SAP) over the 16-bit I/O channel. SPC operates as follows:

- The Compile program will first ask for a source code file name. The source code file always starts with "$".
- SPC will get the file (if it exists) and begin to look for instruction lines. Lines without a semicolon (;) will be ignored as comment lines.
- Lines with a semicolon (;) will be translated into the appropriate binary values.
- If the compiler cannot recognize the source code, an error message will be printed out and the translation will go to the next line and continue. The line which was in error will be printed out for the operator's benefit.
- If no errors occur, the program will indicate a successful compilation and other pertinent data such as the number of code lines, the binary file size, etc.
- The SPC program will enter the binary file in the directory for future reference by the programmer and store the object code on the operator's private disc under the same name as the source code, except for the first character which will be changed to a "@".

Systolic Processor Execution (SPX)

Once the SPC program has finished the binary file, SPX is called to send the file to the systolic array processor (SAP). SPX does the following:

- Get the requested binary instruction file from the disc.
- Start at the first instruction, look for a macro or host procedure instruction. When a macro or host procedure is found, send the previous single word host interface instructions to the SAP using the DMA I/O routine.

- o Perform the requested macro or host procedure.
- o Repeat the process of checking for macros or host procedures until the instruction pointer equals the total number of instructions in the file. (The number of instructions is placed in the first word of the binary file.)
- o When the requested file has been executed, the operator can exit SPX, execute the file again, or get another file to execute.

Miscellaneous Programs

There are also a few programs which are used for special purposes not directly associated with programming the array to perform algorithms. These are:

- o Sytolic Processor EPROM (SPE): This program enables the operator to program the EPROM used in the systolic processor element.
- o Systolic Processor Test (SPT): This program sends test programs to the array and logs errors over long periods.
- o Systolic Program Listing (SPL): This program lists the systolic program source code in a slightly different manner than the Hewlett-Packard utility programs. SPL uses a period in column one to indicate that a new page is desired.

SYSTOLIC ARRAY PROCESSOR INSTRUCTIONS

The systolic array is controlled via a 16-bit data channel coming from the HP 1000 system. The commands and data for the systolic array are sent over this channel and 8-bit data are read back by the HP.

The software for the systolic array will be generated in a source file consisting of the instructions described below. These instructions are made up of mnemonics, hexadecin ' operands and characters. The compiler program SPC converts the source file into binary files which are sent to the array over the 16-bit I/O channel.

There are two types of instructions: (1) single word instructions, which the host interface logic interprets, and (2) macro instructions (groups of single word instructions), which may also require other action by the host.

Single Word Instructions

All operations of the systolic array are controlled by 32 commands which the logic of the systolic array's host interface interprets, one by one. The interface logic determines which of the 32 commands is being sent by examining the lower 5 bits of the upper byte of the 16-bit input channel. The lower byte contains zero, one or two operands depending on the instruction. Refer to the host command table in Appendix A for a complete description of the commands.

Macro Instructions

There are ten macro instructions. The first, "CA" causes a disc resident group of single word instructions to be read from the disc file and inserted in line with other single word instructions. This is a very powerful instruction since there can be a large number of files on the disc to perform any operations necessary.

The next eight macros are used for output and input of data from the data buffer in the HP to the dta buffer in the SAP. Data flow direction is relative to the SAP. The formats of these macros are shown below.

FORMAT	DESCRIPTION
CA @-----	CALL THE INSTRUCTION FILE NAMED @----- AND INSERT AT THIS LOCATION IN THE OBJECT CODE.
IA #-----	GET DATA FILE #-----: PUT IN BUFFER BLOCK A; LOCATIONS 1 THRU 256
IB #-----	GET DATA FILE #-----: PUT IN BUFFER BLOCK B; LOCATIONS 257 THRU 512
IC #-----	GET DATA FILE #-----: PUT IN BUFFER BLOCK C; LOCATIONS 513 THRU 768
ID #-----	GET DATA FILE #-----: PUT IN BUFFER BLOCK D; LOCATIONS 769 THRU 1024
OA #-----	STORE DATA IN LOCATIONS 1 THRU 256 IN DATA FILE NAMED #-----
OB #-----	STORE DATA IN LOCATIONS 257 THRU 512 IN DATA FILE NAMED #-----
OC #-----	STORE DATA IN LOCATIONS 513 THRU 768 IN DATA FILE NAMED #-----
OD #-----	STORE DATA IN LOCATIONS 769 THRU 1024 IN DATA FILE NAMED #-----

The tenth macro instruction causes the host to execute a particular subroutine within the host. The action taken by the subroutine can be anything at all including getting other files of instructions and outputting them to the systolic array.

The format of this instruction is:

HP XXXXXX

The two characters "HP" are followed by the six (maximum) character subroutine name. (In the future it may be useful to be able to pass other data from this instruction, e.g., character strings, buffer names, etc., but for now, only the subroutine name will be passed.)

Examples of the HP (host procedure) instruction are:

```
HP DMRRCC  DOUBLE PRECISION INTEGER DISPLAY OF SPE REGISTERS.
HP FMRRCC  FLOATING POINT DISPLAY OF SPE REGISTERS.
HP LOADX   LOAD STORE "X" (A-B-C) FROM THE BUFFER INTO THE ARRAY.
HP READX   READ THE ARRAY "X" (A-B-C) INTO BUFFER BLOCK "X".
HP PAUZXX  PAUSE.  ASKS OPERATOR TO CONTINUE OR TERMINATE.
HP STEPXX  PRINTS "STEP XX" ON LUPRNT.
HP WAITXX  WAITS FOR XX TENTHS OF A SECOND.
HP !NNLL   PRINTS REMARK !NN IN SAP SOURCE FILE, LL LINES.
HP TSTART  START TIMING AN OPERATION.
HP TSTOP   STOP TIMING THE OPERATION AND PRINT OUT TIME DIFFERENCE.
HP SPORTX  S-PORT READ, 8 BYTES ARE READ FROM THE INTERFACE S-PORTS.
HP SPDATA  DISPLAY THE 8 X 8 S-PORT DATA ARRAY.
HP @XXXXX  FETCH AND EXECUTE INSTRUCTION FILE @XXXXX.
```

DMRRCC

The DMRRCC routine prints out the requested arrays on the enabled terminal using double precision integer (32-bit) format.

The format for the DMRRCC macro is as follows:

```
D  - CALLS THE DOUBLE PRECISION INTEGER DISPLAY MACRO
M  - MODE: Ø = NO DISPLAY          1 = A ARRAY ONLY
           2 = A & B ARRAYS        3 = A, B & C ARRAYS
           4 = B ARRAY             5 = C ARRAY
           6 = D ARRAY             7 = SPARE
RR - FIRST ROW AND LAST ROW.  R = 1 THRU 8.
CC - FIRST COLUMN AND LAST COLUMN.  C = 1 THRU 8.
```

FMRRCC

The FMRRCC routine prints out the requested arrays on the enabled terminal, using floating point format.

The format of the FMRRCC macro is as follows:

F - CALLS THE FLOATING POINT DISPLAY MACRO

M - MODE: Ø = NO DISPLAY 1 = A ARRAY ONLY
2 = A & B ARRAYS 3 = A, B, & C ARRAYS
4 = B ARRAY 5 = C ARRAY
6 = D ARRAY 7 = SPARE

RR - FIRST ROW AND LAST ROW. R = 1 THRU 8.

CC - FIRST COLUMN AND LAST COLUMN. C = 1 THRU 8.

LOADX

The LOADX routine moves the data in the "X" block (X=A,B,C) of the SAP buffer into the "X" store of the array, i.e., into every processing element. This is a complete 8 x 8 matrix load.

READX

The READX routine moves the data in the "X" store of every array element into the "X" block of the SAP buffer. This is a complete 8 x 8 matrix read, i.e., every element is read.

PAUZXX

The PAUZXX routine stops the program and prints "PAUSE XX -- CONTINUE OR ABORT? [GO-AB] - " on the CRT. This enables the operator to continue or abort the program.

STEPXX

The STEPXX routine prints "STEP XX" on the output terminal (CRT or printer).

WAITXX

The WAITXX routine delays the program execution for XX tenths of a second. This is to allow viewing of the LEDS on the ARRAY or CRT printouts at a particular step in the program.

HP !NNLL

This macro is used for printing out messages on the enabled terminal. There can be from 1 to 99 messages, each being from 1 to 99 lines long. The message is written in with the source code. The host will go to the disc and search through the source code file for the requested message. An example is shown next.

```
**********
!Ø3
   THIS IS THE FIRST LINE OF THE MESSAGE.
   THIS IS THE SECOND LINE.
   THIS IS THE LAST LINE.
HP !Ø3Ø4; DISPLAY EXAMPLE MESSAGE - 4 lines.
**********
   FOUR LINES WILL BE PRINTED ON THE ENABLED TERMINAL.
```

HP TSTART - HP TSTOP

These macros must be used in pairs. The HP TSTART is placed at the beginning of the code to be timed. The host real time clock is read and stored. The HP TSTOP macro reads the host real time clock, subtracts the starting time from the stop time and displays the difference on the enabled terminal.

HP SPORTX - HP SPDATA

These macros are used to read the S-PORTS of the SPES. The single word host interface instruction "RS" must be used to put the interface into the mode to output the 8 S-registers in the interface to the host. The user must provide the other instructions to move the S-PORT data he wants into the interface S-registers before using the "RS" instruction. The "X" in the HP SPORTX call specifies in which column of the S-PORT data array to store the data being read. When the 64 S-PORTS have been read, if that is desired, the 8 x 8 array of S-PORTS can be displayed on the enabled terminal by using the HP SPDATA macro.

HP @XXXXX

This macro fetches the binary instruction file @XXXXX and executes it. This must be the last instruction in a file since the new file overwrites the current file.

PROGRAMMING THE SYSTOLIC PROCESSOR TESTBED

The source code for the systolic array processor (SAP) will be written using the Hewlett-Packard editor. Source code will, therefore, be generated using a familiar tool which will allow generous commenting in the body of the program to ensure easy reading and documentation.

The source code will use the two types of instructions discussed earlier, i.e., single word instructions and macro instructions. The rules for source code writing are as follows:

1. Each line of text is either a comment or an instruction group.
2. An instruction line must have a semicolon (;) following the last or only instruction. Lines without a semicolon are treated as comment lines.
3. When several instructions appear on a line, they must be separated by commas (,).
4. The instructions will use a space ( ) to separate operands from the mnemonic.
5. Comments may follow the semicolon (;) on a command line. The translator will ignore anything beyond the semicolon.

The following is an example of a SAP source:

```
$XAMPL - EXAMPLE PROGRAM
-------------------------------------------------------------------------------
    THIS IS A TEST PROGRAM TO CLEAR THE SAP DATA REGISTERS, LOAD THE B
REGISTERS WITH THE DATA FILE #LRC# AND DISPLAY THE FULL A,B AND C ARRAYS.
-------------------------------------------------------------------------------

CI, CP, DY Ø4;  CLEAR THE INTERFACE, THE SPES AND DELAY 64 MICROSECONDS

CA @CLR;  CALL THE MACRO @CLR TO CLEAR ALL THE A, B AND C-PORTS.

IB #LRC#;  GET DATA FILE #LRC# AND PUT IT IN LOCATIONS 257 THRU 512 IN
           THE SAP BUFFER.

HP LOADB;  MOVE THE SAP BUFFER B DATA BLOCK INTO THE B ARRAY.

HP D31818;  DISPLAY THE A, B AND C ARRAYS ON THE CRT.

THIS COMPLETES THE PROGRAM.

END
```

SUMMARY

The systolic array processor testbed has been fabricated to aid in the development of algorithms and architectures for high speed signal processing of communications and undersea surveillance signals.

The main goal in the design of the testbed has not been speed, but flexibility. (It is well known that general hardware is always slower than special hardware when solving a specific problem.)

Currently the testbed is being used at NOSC to implement algorithms being developed elsewhere, with the intention of discovering the shortcomings and advantages of the algorithms and proposed hardware.

NOSC extends the invitation to any interested parties to consider using this testbed via telephone lines to perform their own investigations into the systolic array technology.

APPENDIX A

A SYSTOLIC ARRAY PROCESSOR IMPLEMENTATION

This article appeared in the Proceedings of the SPIE International Technical Symposium, San Diego, CA., 25-28 August 1981, Vol. 298.

A Systolic Array Processor Implementation

J. J. Symanski

Naval Ocean Systems Center
San Diego, California 92152

Abstract

A combination of systolic array processing techniques and VLSI fabrication promises to increase signal-processing capabilities by a factor of 100 or more. To achieve a timely marriage of algorithms and hardware, both must be developed concurrently. This article describes the hardware for a programmable, reconfigurable systolic array testbed, implemented with presently available integrated circuits and capable of 32-bit floating-point arithmetic. While this hardware presently requires a small printed circuit board for each processing element, in a few years one or two custom VLSI chips could be used instead, yielding a smaller, faster systolic array processor. This testbed will aid in the evaluation of the many parameters which will have to be optimized in order to design these custom chips.

Introduction

The systolic array concept involves the inherent high throughput and simplicity offered by a lattice of identical processing elements, all operating in parallel on data synchronously flowing through the structure. The prospects for fabricating an array element (or several elements) on a single chip appear very good. However, many details of the algorithms, data flow, control, input/output, numerical accuracy, speed, etc, have to be determined before a particular chip design can be undertaken.

The goal of this work is to build a systolic array testbed which is flexible enough to allow experimentation with algorithms and configurations so that intelligent decisions can be made when it comes time to specify the chip architecture for a particular set of applications. The hardware implementation is shown conceptually in Figure 1. The array consists of a cabinet containing a 8-by-8 array of systolic processor circuit boards, a mother board, and a rack for the host-interface electronics.

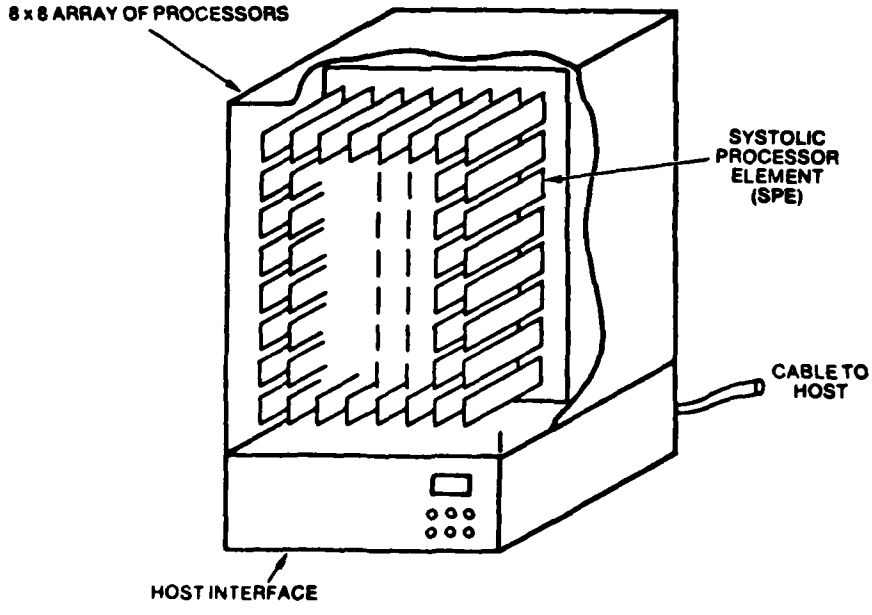


Figure 1. Systolic array configuration.

Design history and rationale

In the initial design of this testbed, many questions regarding tradeoffs and decisions had to be answered. For instance, should we use bit-serial or bit-parallel computation? what dynamic range is necessary for a useful processor? what speed of computation is reasonable? how complex or smart should each processor be? how many elements will be reasonable to connect? and what will be the packaging approach? Only the final design will be presented here, without describing the many possible alternatives.

Since the overriding concern in this testbed is for flexibility to allow experimentation, it was decided to use a microprocessor with EPROM and RAM to allow the maximum programmability in the systolic processing element.

As for the dynamic range, comments from prospective users led us to settle on a 32-bit floating-point capability. Here a tradeoff between software and hardware implementation led to the use of an Arithmetic Processing Unit (APU) or peripheral processor (Intel-8231 or AMD-9511).

Bit-serial computation was considered as a possibility because of its expandability and low pin count. However, the need for wide dynamic range and the relative complexity and long design time a bit-serial design would require, made the bit-parallel approach of the APU more attractive.

The type of communication path to the SPE (serial or parallel) effects the speed of operation and the hardware required for an array. Serial communication was found to be more advantageous for several reasons. Since each SPE has six I/O ports, an eight-bit parallel path would require 48 pins and 48 driver/receiver buffers in each SPE. In addition, as in VLSI design, interconnection could become a major problem. Furthermore, to obtain flexibility in the intercommunication of the array, multiplexers have been placed in some of the data paths. The use of parallel communication would have significantly increased the amount of hardware required.

After the basic processor complexity was determined, the system was partitioned into a large (approximately 23-by-16-inch) mother board with each systolic processor on a separate 2.5-by-8-inch printed circuit board, mated to the mother board by using an edge card connector on a short side (see figure 1.)

Systolic array testbed system

The systolic array testbed system is composed of a minicomputer system interfaced to the array of systolic processor elements (SPEs). The host is an HP-1000 minicomputer with the usual compliment of printer, disk storage, keyboard-CRT, etc. The systolic array is housed in a cabinet approximately 28 by 19 by 21 inches. The interface circuitry uses a single 16-bit data path from the host HP-1000 to communicate data and commands to the array.

Commands and data are generated in the host by the operator, using interface programs written in FORTRAN. Algorithms can be conceived, put into a series of commands for the systolic array processor, and tested for validity. Data computed in the array can be read by the host HP-1000 minicomputer and displayed for the operator.

Systolic array testbed architecture

Many other papers have discussed the theoretical aspects of the systolic array's communication of data and other properties. We have implemented the original H. T. Kung architecture, as shown in Figure 2a (ref 1: Mead and Conway). Then, by substitution of squares for hexagons, appropriate rotation of the communication paths, and realignment of the processors on a square grid, we have Figure 2b. Now the A data paths are horizontal, the B paths are vertical, and the C paths are along a diagonal.

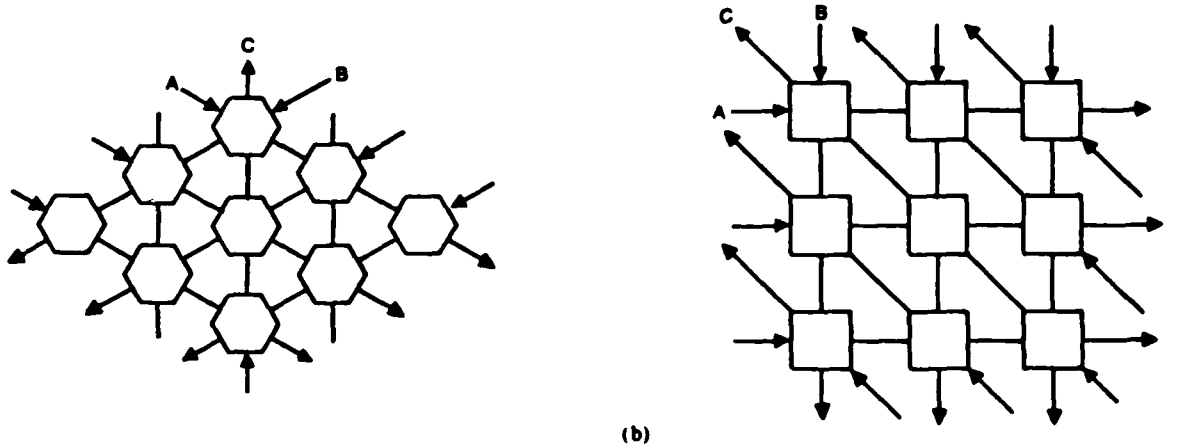


Figure 2(a-b) The transformation from hexagonal to square structure.

In this implementation, there are also virtual rows and columns along the edges of the array, as shown in Figure 3. These virtual rows and columns perform the interfacing between the parallel data path from the host and the serial communications of the systolic array processing elements. These virtual rows and columns can be thought of as existing on either side of the array, since the data paths from the SPEs on the "far side" of the array wrap around and are also connected to the A, B, or C rows and columns.

To achieve architectural flexibility, the serial data paths from the virtual rows and columns pass through multiplexers so that several options for data flow are possible. The data path can be selected in real time by the host processor.

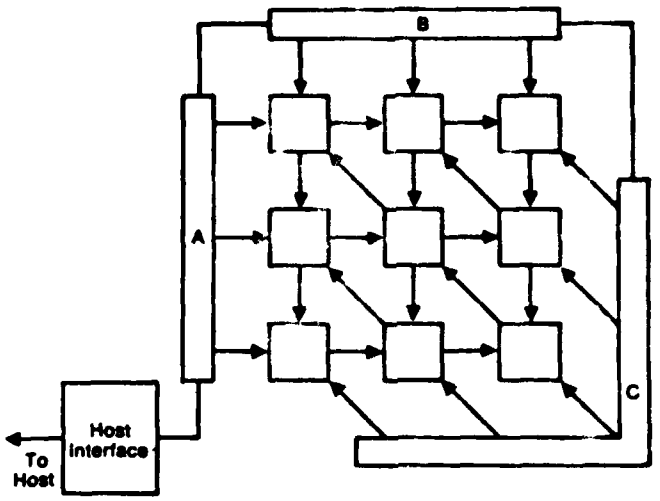


Figure 3. The square systolic array with virtual row and columns.

Array operation

The main reason for fabricating this testbed is to develop the skills for designing and using systolic processors. The control of the data flow into, out of, and inside of the array is under the control of the operator. The interface circuitry responds to the commands listed in Table 1. By using these commands and routines programmed into the SPE, the operator can develop algorithms to perform matrix operations such as matrix-vector multiplication, convolution, the discrete Fourier transform (DFT), matrix-matrix multiplication, the L-U decomposition of matrices by Gaussian elimination, matrix inversion, etc.

The use of a general-purpose minicomputer as the driver for the systolic array and a microprocessor in the systolic processing element gives us unlimited flexibility in developing algorithms. Through the use of interface routines, algorithms can be tried, evaluated, changed, and tried again in a few minutes. Furthermore, in cases where the output must be manipulated and fed back into the array, the manipulation of the data can be done in the host by using the high-order language capability.

However, the operating speed of the systolic array is lowered because of the minicomputer operating system's characteristics, which result in millisecond delays in sending data or commands to the array. It would also be more efficient for the host to pass only high-level instructions to the array instead of the many individual commands. It is expected that, in using the array to perform algorithms, modules of commands will be developed. Examples are sequences of commands for data formatting, array loading, array operations such as matrix multiplication, etc. In the future, a dedicated microprocessor front end could be used to operate the array by using these modules of commands with only the high-order instructions and data coming from the host. This would significantly increase the operating speed of the array as well as ease the programming task for the operator.

While this particular test-bed hardware may not be the most efficient or fastest for implementating a particular matrix operation, it will be a flexible tool which will give us valuable experience in developing architectures and algorithms needed for optimized solutions in the future.

The host interface

Figure 4 shows a block diagram of the host interface. Commands for the array and data are sent to the interface and interpreted by the interface control logic. The list of commands is shown in Table 1. Control lines and system clocks are generated as needed and sent to the array. The SPE commands are single-byte words broadcast in rows to the entire array. (A single command placed the the S register of the SPE may initiate many instructions in the SPE.) Status or other data can be read back on the same lines.

The data buffer can be written into or read from in various ways, depending on the commands sent to the interface. Once data have been sent from the host, subsequent commands read the data out of the buffer and load them into the parallel-to-serial registers which make up the virtual rows and columns A, B, and C. Other commands to the interface move the data, in bit-serial format, into the array of SPEs. When computation is complete, data are shifted out of the array into the virtual rows and columns and then moved in parallel format into the buffer. The host can then read the results and display them for the operator.

Table 1. Host Interface Commands

Mnemonic	Description
LS *	Load Single system register and increment to next register
LA *	Load All system registers with this function code
RS	Read System registers (8 words)
BA *	Buffer Address (bits 0-7)
BB *	Buffer Block (bits 8-11)
BE *	Buffer Enable (input/output selection)
DL	Load Data into register from buffer and increment
DR	Read Data register into buffer and increment
MO	Move data and system registers (One clock)
MS	Move System register (8 clocks)
MD	Move Data registers (8 clocks)
MB	Move Both data and system registers (8 clocks)
SP *	Select Processor multiplexer (two LSBs)
SS *	Select S controls (input mux, register controls)
SA *	Select A controls (input mux, register control)
SB *	Select B controls (input mux register control)
SC *	Select C controls (register control)
SR *	Select Register (A, B, CA, CB)
AC	Alternate Clock (enables external DCLK)
IT	InTerrupt SPE
IW	Interrupt SPE and Wait for ready
CP	Clear Processors (reset SPE)
WT	WaiT for Ready
LR *	Load Row register with lower byte
LC *	Load Column register with lower byte
SL	SLow processor clock
FS	FaSt processor clock
DY *	DelaY (# of microseconds in lower byte: range is 0 to 255)
CI	Clear Interface
NO	No Operation

*Operand required.
Note: MSB of command byte determines which array (real or imaginary) gets command.

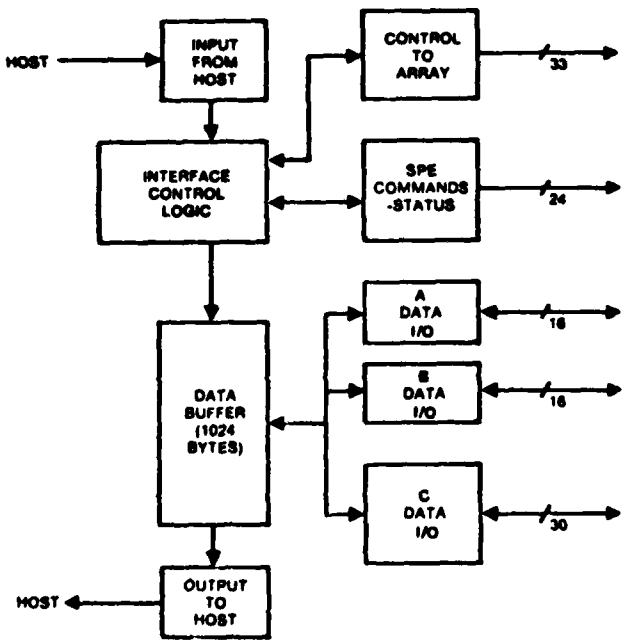


Figure 4. Systolic array host interface block diagram.

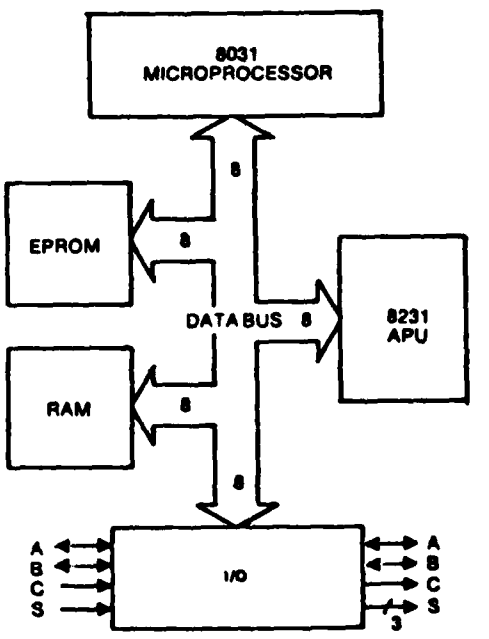


Figure 5. Systolic processor element block diagram.

The Systolic Processor Element (SPE)

The block diagram for the systolic array processing element is shown in Figure 5. The microcomputer used is the Intel 8031. This device was chosen for its speed, internal RAM, many I/O pins, and ease of bit and byte manipulation. The data moves about in the SPE over a multiplexed 8-bit data/address bus.

Computation is accomplished in the Arithmetic Processing Unit (APU). This unit is capable of several formats (16-bit fixed point, 32-bit fixed point, and 32-bit floating point). Many operations are available such as add, subtract, multiply, divide, square root, several trigonometric functions, etc. The device comes in a 24-pin package.

The EPROM and RAM are standard devices. The EPROM is 4k X 8 bits. The RAM is 1k X 8 bits. The EPROM is used to store routines which perform data manipulation. This gives the system a hierarchical approach so that a single byte transmitted to the processor initiates a sequence of operations. The RAM can be used for data storage. This results in a "3rd dimension" of matrix storage. The RAM can also be used for storage of programs during algorithm development. Once the algorithms have been perfected, they will be put into EPROM.

The I/O from the processor is bit serial. The main reason for serial I/O is to minimize pinouts and driver/receiver requirements. Four 299 universal 8-bit parallel/serial shift registers constitute the I/O ports. The I/O registers are loaded and read by the microprocessor under program control.

Figure 6 shows the SPE in somewhat greater detail. This is not a complete schematic because of the complexity of the processor. The SPE consists of 18 integrated circuits, mostly 7400-series low-power Schottky TTL devices. The 8031 microcomputer is the overall controller of data flow. The interrupt (SINT) and reset (SRST) inputs to the 8031 are the means by which the interface circuitry "gets the attention" of a processor. The 8282 is an 8-bit latch to demultiplex the lower 8 bits of the memory address from the data bus. The 2732 is the 4k X 8-bit EPROM. The 2114 (two devices) make up the 1k X 8-bit RAM.

The 8231 (APU) obtains its data over the 8-bit data bus from the microcomputer. The APU uses a 4-MHz clock. The 8031 reads a status word from the APU to determine when the answer is available.

The I/O registers are also on the 8-bit data bus. The 138 3-line-to-8-line demultiplexer takes in the address bits A14, A13, and A12, and selects one of the I/O ports, RAM or EPROM. Address bit A15 is used to select the APU. The microcomputer also outputs control lines which control the I/O registers; i.e., shift right, shift left, load, or read.

The 153 at the bottom of the schematic is used to select a particular processor or group of processors to perform a task different from other SPEs in the array during a particular cycle. There are four selection modes which are determined by the processsor select (PS0 and PS1) inputs. The first is all processors. The second is a particular column of processors. The third is a particular row of processors. The fourth is a single, particular processor. The values of PS0, PS1, ROW, and COLM are determined by inputs to the interface from the host. When the conditions are met in a particular processor, the system outputs (SR and SC) are enabled as well as the system register clock.

The A, B, and C registers are used primarily for data. The S (or system register) is used for control purposes. An 8-bit word can be sent from the interface logic to all or selected processors via the S register. This 8-bit word will be interpreted by the 8031 microcomputer as a command to perform a specific task.

Future plans

The systolic array testbed will be used to gain insight into the advantages of and problems encountered in constructing systolic processors and utilizing the systolic processing approach for matrix operations. The experience gained in implementing a wide variety of algorithms with this testbed will have benefits in both algorithm and hardware design. In the process of trying to create efficient, fast, understandable algorithms, we will also find the capabilities that the hardware should have in data paths, communications of data and commands, processing power, memory, etc. This experience will enable the timely marriage of optimized algorithms and hardware in solving complicated signal-processing problems with VLSI. In 2 or 3 years, a single chip instead of the present printed circuit board, could be designed to communicate in 32-bit numbers and perform 32-bit floating-point mathematical operations in a few microseconds instead of the 200-300 microseconds required by the present implementation. Control and input/output concepts will also be required for efficient utilization of VLSI chips in a broad range of applications. This testbed will provide insight into many of these problems.

Acknowledgements

The author wishes to thank Keith Bromley and Harper J. Whitehouse for their many suggestions and guidance in the design of the array architecture. This work has been supported by funding provided by the Naval Ocean Systems Center Independent Research/Independent Exploratory Development (IR/IED) program.

Reference

1. Mead, C., and Conway, L., Introduction to VLSI Systems, Addison-Wesley, 1980, pp. 271-292.

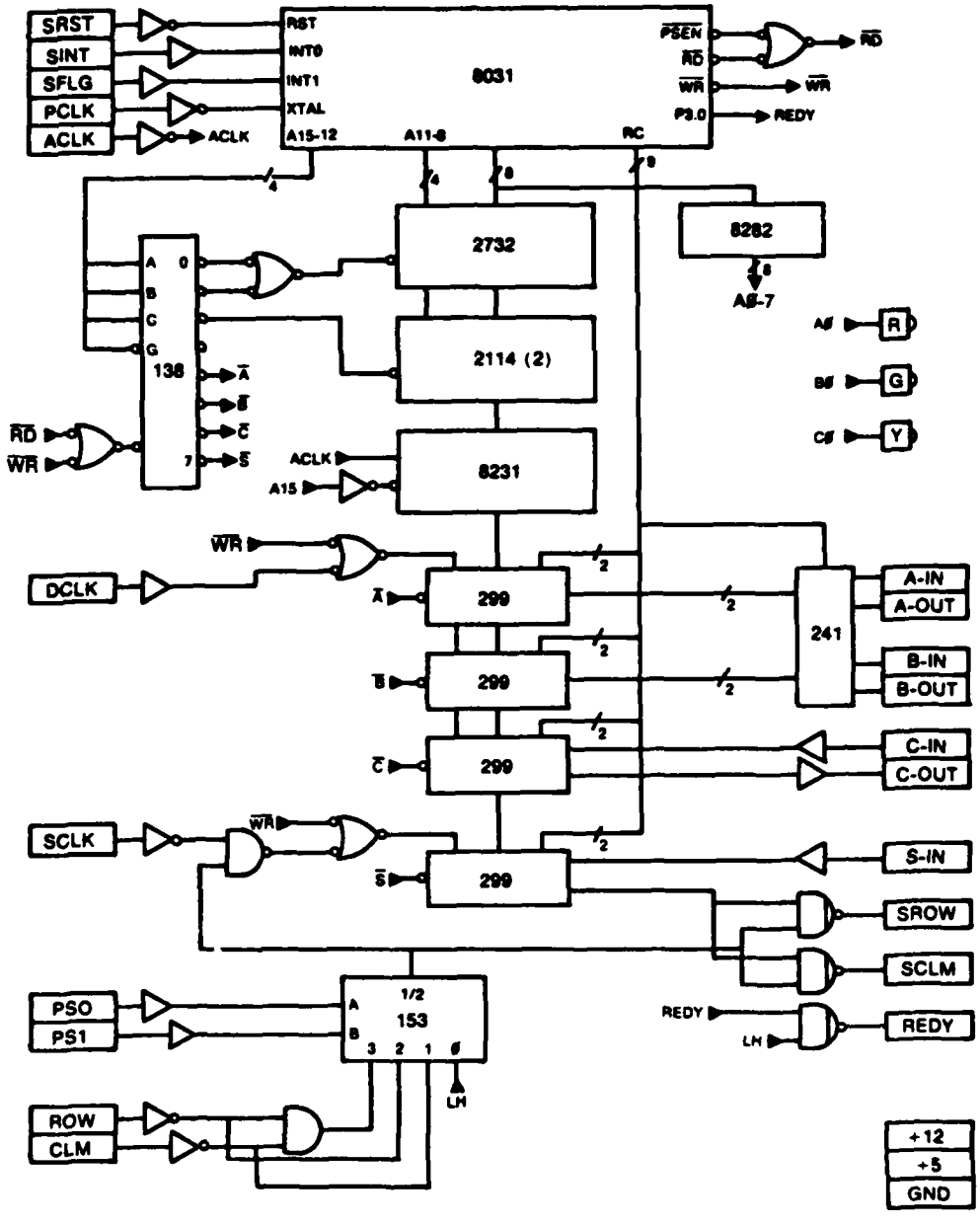


Figure 6. Systolic processor element.

APPENDIX B

PROGRESS ON A SYSTOLIC PROCESSOR IMPLEMENTATION

This article appeared in the Proceedings of the SPIE International Technical Symposium, San Diego, CA., 25-28 August 1981, Vol. 298.

Progress on a Systolic Processor Implementation

J.J. Symanski

Naval Ocean Systems Center, San Diego, California 92152

Abstract

Parallel algorithms using systolic and wavefront processors have been proposed for a number of matrix operations important for signal processing; namely, matrix-vector multiplication, matrix multiplication/addition, linear equation solution, least squares solution via orthogonal triangular factorization, and singular value decomposition.

In principle, such systolic and wavefront processors should greatly facilitate the application of VLSI/VHSIC technology to real-time signal processing by providing modular parallelism and regularity of design while requiring only local interconnects and simple timing.

In order to validate proposed architectures and algorithms, a two-dimensional systolic array testbed has been designed and fabricated. The array has programmable processing elements, is dynamically reconfigurable, and will perform 16-bit and 32-bit integer and 32-bit floating point computations. The array will be used to test and evaluate algorithms and data paths for future implementation in VLSI/VHSIC technology.

This paper gives a brief system overview, a description of the array hardware, and an explanation of control and data paths in the array. The software system and a matrix multiplication operation are also presented.

Introduction

The systolic concept involves the inherent high throughput and simplicity offered by a lattice of identical processing elements, all operating in parallel on data flowing through the structure.[1,2] The prospects for fabricating an array element (or several elements) on a single chip appear very good. However, many details of the algorithms, data flow, control, input/output, numerical accuracy, speed, etc. have to be determined before a particular chip design can be undertaken.

The goal of this work is to build a systolic array testbed which is flexible enough to allow experimentation with algorithms and configurations so that intelligent decisions can be made when it comes time to specify the chip architecture for a particular set of applications. The testbed was not designed for optimum architecture for a specific application, but for flexibility in evaluation of various algorithms and architectures which may be implemented in the future. There are input/output limitations due to the single I/O channel from the host system (a common problem for array processors) as well as the speed limitation set by the relatively slow arithmetic processing unit.

System Overview

The systolic array testbed system is composed of a minicomputer system interfaced to the array of systolic processor elements (SPEs). The host is a minicomputer with the usual complement of printer, disc storage, keyboard-CRT, etc. The systolic array is housed in a cabinet approximately 28 by 19 by 21 inches. (See Figure 1.) The interface circuitry uses a single 16-bit data path from the host minicomputer to communicate data and commands to the array.

Commands and data are generated in the host by the operator, using interface programs written in FORTAN or PASCAL. Algorithms can be conceived, put into a series of commands for the systolic array processor, and tested for validity. Data computed in the array can be read by the host minicomputer and displayed for the operator.

Many other papers have discussed the theoretical aspects of the systolic array's communication of data and other properties.[1,3,4,5] We have implemented the original H. T. Kung hexagonal interconnect architecture, as shown in reference 2. By substitution of squares for hexagons, appropriate rotation of the communication paths, and realignment of the processors on a square grid, we have the square array. Now the A data paths are horizontal, the B paths are vertical, and the C paths are along a diagonal.

In this testbed, there are virtual rows and columns along the edges of the array as shown in Figure 2. These virtual rows and columns perform the interfacing between the parallel data path from the host and the serial communications of the systolic array processing elements. These virtual rows and columns can be thought of as existing on either side of the array, since the data paths are just 8-bit shift registers connected in a circular manner.

To achieve architectural flexibility, the serial data paths into the array pass through multiplexers so that several options for data flow are possible. The data path can be selected in real time by the host processor. The various configurations and their uses are discussed in reference 4.

Figure 3 shows a block diagram of the host interface. Commands and data for the array are sent to the interface and interpreted by the control logic. Control lines and clocks are generated and sent to the array as appropriate.

The SPE block diagram is shown in Figure 4. The microprocessor is an Intel 8031. The RAM and EPROM are standard devices. The Arithmetic Processing Unit (APU) is the Intel 8231. The I/O section contains four universal 8-bit shift registers. The A, B and C registers are for data. The S register is used to control the operation of the SPE. (See references 3 and 4.)

Figure 1. Systolic array hardware.

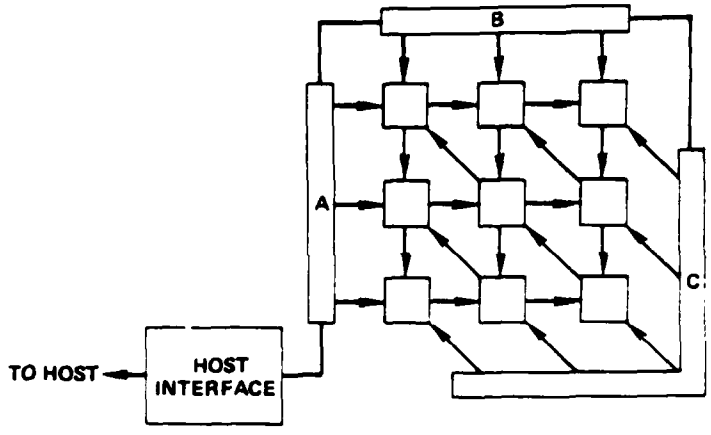


Figure 2. The square systolic array with virtual rows and columns.

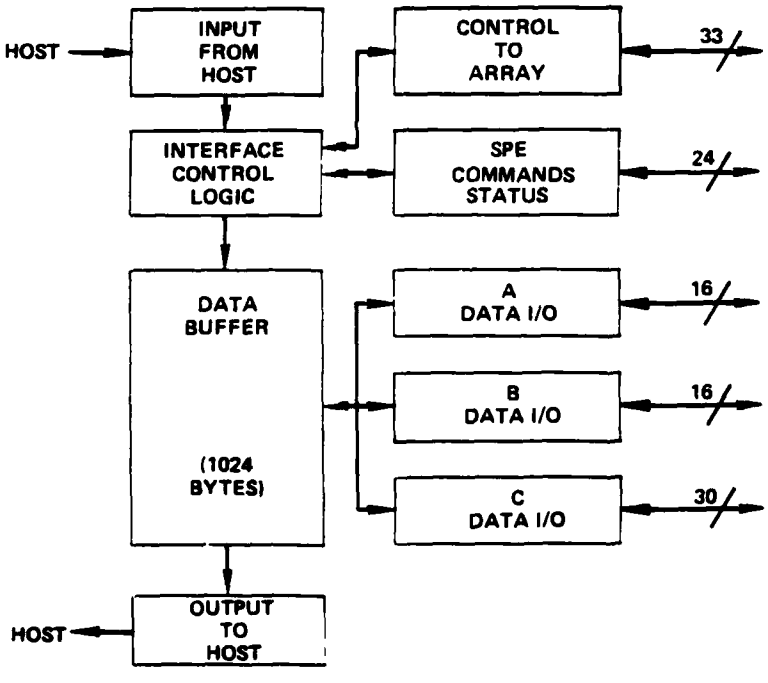


Figure 3. Systolic array host interface block diagram.

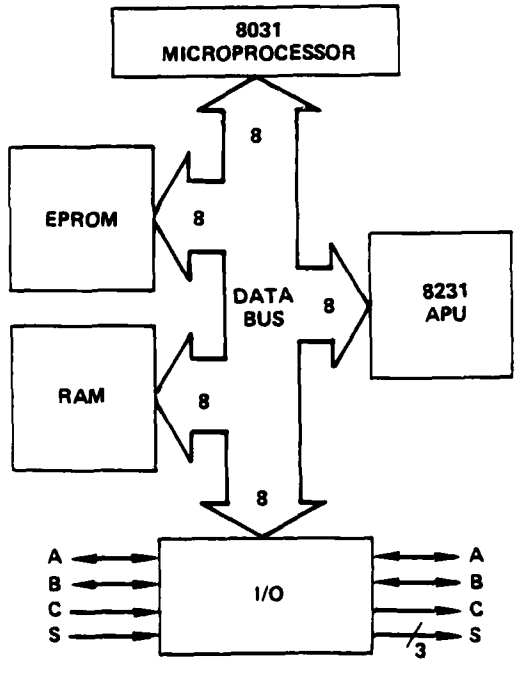


Figure 4. Systolic processor element block diagram.

It is important to note that the flexibility of this testbed is obatined through the use of a microprocessor in the SPE, thus allowing us to interchange the roles of A, B, and C at will.

Array Hardware

The assembled 8-by-8 array, with the interface electronics, is shown in Figure 1. The SPEs mount directly onto the 16 x 23 inch motherboard with edgecard connectors. All signals to the array, from the interface circuitry in the rack below the array, are carried by four flat cables. The power supplies are mounted behind the array. The interface circuitry consists of four wire-wrap boards with about 60 TTL ICs on each board.

Layout of circuitry can be a problem in printed circuit board design as well as VLSI, but is usually not as critical. Also, it is always good practice to make the layout as regular and logical as possible. This is one of the strong points of systolic arrays; i.e., the regularity of interconnections. Figure 5 shows a 2-by-2 portion of the array. Note the regularity of the pattern. There are 120 signal lines from the interface circuitry to the array. Signal lines were made as wide as possible and ground planes were used generously. Two ground returns are used, one for power and one for signal.

The SPE board is shown in Figure 6. It is a four layer board. The two internal layers are used for power and ground only. Here again, there are two ground patterns, one for the return of power and one for signal grounding and noise shielding.

There are 18 chips on the SPE board. The most expensive parts, the Arithmetic Processor ($100), the 8051 microprocessor ($18), and the 2732 EPROM ($12), are on sockets. The other ICs are standard TTL devices ($22 total). The printed circuit board costs about $25, not including the cost of layout. Total cost for the SPE components is about $190.

Assembly of the system has had the usual problems which show up when putting together a complex system. The SPE was fabricated in wire-wrap form in order to check its operation and develop code for the 8051 microprocessor. This was done on a separate system independent of the array host and interface system. Then the array interface was assembled and interfaced to the host. Software was used to verify the interface operation before SPEs were connected to the array. Then a wire-wrapped 3-by-3 array of SPEs (printed circuit board prototypes) was connected to the interface electronics. Software was again used to verify operation of the whole system. Finally the full 8-by-8 array was assembled. Software programs were generated to test the full array. The array is now fully functional. A matrix multiply implemented on the system will be described later.

Control Flow

The individual processors are identical with respect to hardware and program in EPROM. Referring to Figure 4, the A, and B and C registers are used for data. The S register is used by the host to store a value (8 bits) which is interpreted by the SPE's microprocessor as a command to perform a certain operation. Examples of operations are: move the byte in the A register into memory, multiply the stored values of A and B, change shift direction of the registers, etc. (There are presently about 120 commands for data movement and computations programmed into the EPROM, utilizing about half of the 4K bytes available.) These S registers are connected to the interface as shown in Figure 7 for a 3-by-3 array. Each row of the array has an S register within the interface. The register for a given row broadcasts the same

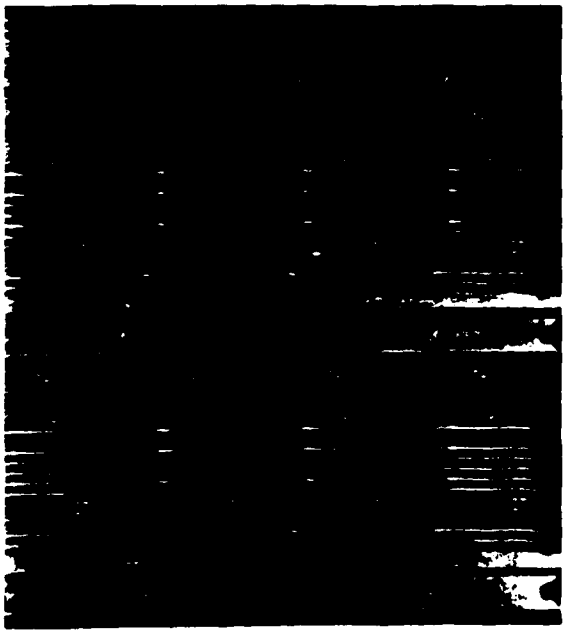


Figure 5. Two-by-two section of array backplane.

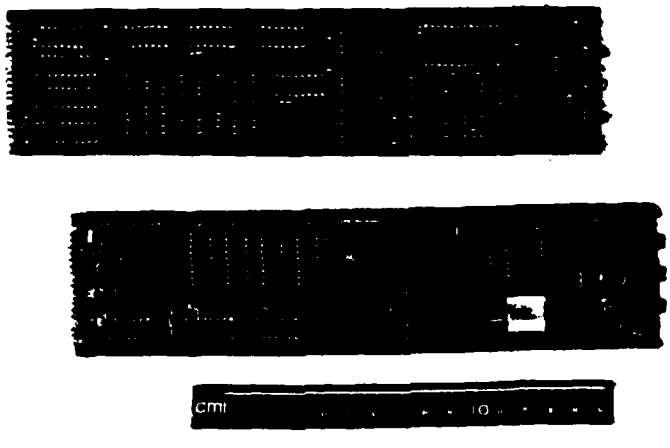


Figure 6. SPE board

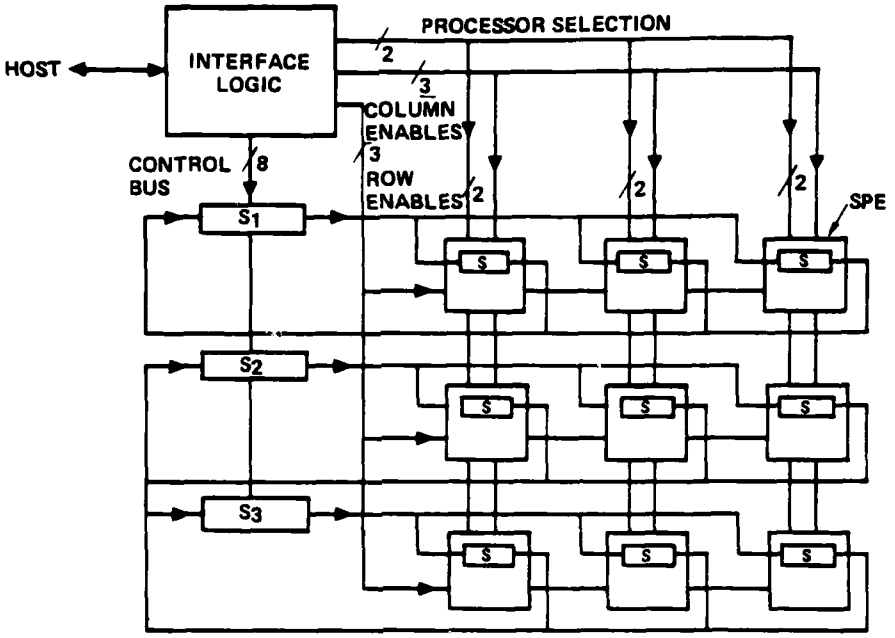


Figure 7. Systolic array control structure.

byte to each S register in that row. The eight S registers within the interface logic can be loaded with the same or different command bytes. Thus, loading rows with different commands will cause different operations in SPEs in different rows.

There is a further possibility for control of individual SPEs called the processor select. On each SPE there are four lines which control the input clock to the S register. Two lines determine the Select Code which controls the 4-to-1 multiplexer which selects one of four inputs. The other two are Row Enable and Column Enable. Table 1 shows the selection possibilities. Select code 3 ANDs the Row Enable and Column Enable inputs to a SPE. Thus, by using the Row and Column select mode, any individual SPE can be selected for an operation different from the operation performed by all the other SPEs.

Note that there is also an output line from the SPE to the registers in the interface logic. These lines are open collector driven to avoid conflicts. With appropriate control, this can be used to pass error status or other information back to the controller and then to the host.

The S register can also be used for other purposes. For instance, all the S registers in the interface can be loaded with the same 8-bit data. With the appropriate routine in EPROM, the same data can be broadcast to the whole array instead of shifting in the data through each SPE. This method can also be used to broadcast new routines to be stored in the SPE RAM for testing. Furthermore, by loading each S register with different values, data can be broadcast to all SPEs along a particular row.

Data Movement

The SPE performs calculations using 32- or 16-bit values. However, all data are moved in bytes to and from the array as well as within the SPE. The host formats the 32-bit values into the host I/O buffer. Once the data have been transferred into the array data buffer, commands from the host move the data from the buffer into the appropriate parallel-to-serial registers in the virtual rows and columns. Subsequent commands shift data into the array as well as within the array. A command byte is put into the S registers of the SPEs and interrupt broadcast to all the SPEs. This causes the microprocessor in the SPE to store the data byte in the appropriate storage location in the SPE RAM. The register load, shift, and data storage cycle is repeated four times for each 32-bit value. Subsequent instructions from the host, using the control structure discussed earlier, initiate computations on the data as desired.

As discussed in reference 4, there are several array configurations possible with this testbed, for instance: linear, square, transpose, dual array, and broadcast configurations. These configurations can be achieved dynamically under the host control. This is done with a multiplexer which selects one of several sources for output from the interface to the left column and also the top row of the array. Table 2 shows the sources for the various configurations. Other sources may be used in the future to enhance the operation of particular algorithms.

Software System

The software system for the systolic array testbed can be thought of in three parts: (1) data entry and display, (2) programming of the array, and (3) execution of the program. The host minicomputer has FORTRAN, PASCAL, and various library programs available. The three functions are described below.

The data entry function will allow matrices of numbers to be generated in several ways: (1) by operator entry, (2) by loading from magnetic tape, and (3) by generation within the host itself. The matrix data generated by one of the above methods are formatted by the

Table 1

Select Code	Selection
0	All SPEs Enabled
1	Rows Enabled
2	Columns Enabled
3	Row and Column Enabled

Table 2. Reconfiguration Multiplexer Inputs and Uses

Input to Left SPE –Row M	Uses
Virtual column – Row M	Data input/output
Right SPE – Row M	Data input/output Column exchange
Right SPE – Row (M-1)	Linear array
Bottom of column M	Transpose
Virtual column – Row 1	Broadcast data
Other array – Row M	Dual array
Other array – Row (M-1)	Dual linear array

host in a manner which allows efficient, speedy transmission to the systolic array processor during program execution, and then stored on disc for future use. A file management system is used to keep track of the various matrices. Also stored with the matrix data is a verbal description of the properties, of, or use for, that matrix.

There will be a structured approach to the programming of the array in that object code modules can be no larger than 256 sixteen-bit words. This forces the programmer to break up long operations into short, easily understood modules. The use of "calls" in the source code enables very long programs to be run with short modules.

Programming the array is done in a manner similar to that used for the host minicomputer. The standard text editor available for writing the FORTRAN and PASCAL programs is used to write the source code for the systolic array. This has the advantage of familiarity and allowing commenting of the systolic processor programs to more easily understand the operations performed by the array. Once source code is written, a compiler program will generate object code for the array, which is a file of 16-bit words sent to the array as instructions. The various programs, or modules, written for the systolic array are also listed in a directory for ready access by the operator.

The execution of programs on the systolic array is accomplished as follows. The operator runs a "main" module which calls several other modules. A "loader" program gets the main module and links it with the called modules as well as any data files required. All the instructions are loaded into a buffer for output to the systolic array or stored for future use as a single operation. The loader will build the program until (a) the buffer size is exceeded, (2) the end of the program, or (3) an input is required. If the program calls for input, the object code up to that point is sent to the array (the array performs the operations) and the host inputs the results from the array. The loader continues in this manner unitl the program is completed. Results received from the systolic array are stored on disc for later display and checking.

Matrix Multiplication

The matrix multiplication implemented uses an in-place accumulation of partial sums.[5] The elements of the array are skewed during input to the array boundary processors. (See Figure 8.) As the rows and columns step through each SPE, the sum of products accumulates in-place.

The A and B matrices are generated in the host by one of the previously discussed methods. The host then formats the data for output to the data buffer in the array interface. Then the host sends commands to the interface logic which moves the appropriate data elements from the data buffer into the peripheral processors of the array. As the program proceeds, commands cause data in each SPE to be multiplied and summed as well as the original data to be moved to the next SPE. When the required cycles of shift data, multiply, and add have been completed, the matrix product of A and B resides in a third C matrix which is simply one of the storage locations of the SPE RAM. (There are 256 such locations available.)

The programming of an 8-by-8 matrix multiplication is relatively straight-forward. However, other operations such as least squares and eigensystem problems, especially in partitioned matrix form (when dealing with matrices larger than the array) may be quite difficult. That is, the selection of data elements for input to the array and movement within the array will require great care. Software tools which show the data residing in each SPE at a particular point in the algorithm will be essential.

In future systems this data formatting and movement will utilize a high speed controller with routines in ROM which will quickly select and move data in and out of the array for a variety of algorithms. Or, more advanced architectures, with greater parallelism in I/O, will be developed to alleviate the I/O bottleneck which can degrade the operation of array processors.

Conclusions

A versatile two dimensional systolic array testbed has been designed, fabricated and programmed to perform matrix operations. While not an optimum architecture for real-world applications, this testbed will allow experimentation with many array configurations and algorithms. The experience gained in implementing a wide variety of algorithms will have benefits in both algorithm and hardware design. In the process of trying to create efficient, fast, understandable algorithms, we will also find the capabilities that the hardware should have in data paths, communication of data and commands, processing power, memory, etc. This experience will enable the timely marriage of optimized algorithms and hardware for solving complicated signal-processing problems with VLSI. In 2 or 3 years, a single chip, instead of the present printed circuit board, could be designed to communicate in 32-bit numbers and perform 32-bit floating-point mathematical operations in a few microseconds instead of the 200-300 microseconds required by the present implementation. Control and input/output concepts will also be required for efficient utilization of VLSI chips in a broad range of applications. This testbed will provide insight into many of these problems.

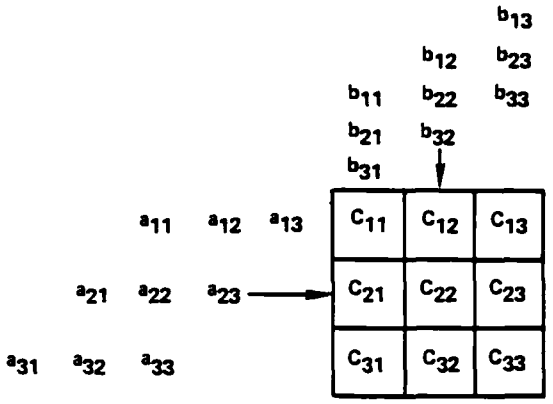


Figure 8. Multiplication of full matrices by an engagement processor.

Acknowledgements

The author wishes to thank Keith Bromley, Harper J. Whitehouse, and Jeffrey M. Speiser for their many suggestions and guidance in the design of the array architecture, John Celto for the SPE processor breadboarding and microprocessor coding, and Dick Martin and Steve Greim for software system assistance.

This work has been supported by funding provided by the Naval Ocean Systems Center Independent Research/Independent Exploratory Development (IR/IED) program and the Naval Electronics Systems Command.

References

1. Kung, H. T., "Why Systolic Architectures?" Computer, Vol. 15, No. 1, Jan. 1982, pp. 37-46.
2. Mead, C., and Conway, L., Introduction to VLSI Systems, Addison-Wesley, 1980, pp. 271-292.
3. Symanski, J. J., "A Systolic Array Processor Implementation," Proc. SPIE, Vol. 298, Real-Time Signal Processing IV, Society of Photo-Optical Instrumentation Engineers, 1981.
4. Bromley, K., Symanski, J. J., Speiser, J. M., and Whitehouse, H. J., "Systolic Array Processor Developments," in VLSI Systems and Computations, H. T. Kung, R. F. Sproull, and G. L. Steele, Jr., (editors), Carnegie-Mellon University, Computer Science Press, Oct. 1981, pp. 273-284.
5. Speiser, J. M. and Whitehouse, H. J., "Parallel processing algorithms and architectures for real-time signal processing," Proc. SPIE, Vol. 298, Real-Time Signal Processing IV, Society of Photo-Optical Instrumentation Engineers, 1981.

APPENDIX C

SYSTOLIC ARRAY PROCESSOR DEVELOPMENTS

This article appeared in the Proceedings of the CMJ Conferences on VLSI Systems and Computation, Carnegie-Mellon University, Pittsburgh, PA., 19-21 October 1981.

Systolic Array Processor Developments

Keith Bromley, J. J. Symanski, J. M. Speiser and H. J. Whitehouse
Naval Ocean Systems Center, San Diego, California 92152

ABSTRACT

The combination of systolic array processing techniques and VLSI fabrication promises to provide modularity in the implementation of matrix operations for signal-processing with throughput increasing linearly with the number of cells utlized. In order to achieve this, however, many design tradeoffs must be made.

Several fundamental questions need to be addressed: What level of complexity (control) should the processor incorporate in order to perform complicated algorithms? Should the control for the processing element be combinatorial logic or a microprocessor? The broad application of a systolic processing element will require flexibility in its architecture if it is to be produced in large enough quantities to lower the unit cost so that large arrays can be constructed.

In order to have a timely marriage of algorithms and hardware we must develop both concurrently so that each will affect the other. A brief description of the hardware for a programmable, reconfigurable systolic array test-bed, implemented with presently available integrated circuits and capable of 32 bit floating point arithmetic will be given. While this hardware requires a small printed circuit board for each processor, in a few years, one or two custom VLSI chips could be used instead, yielding a smaller, faster systolic array. The test-bed is flexible enough to allow experimentation with architecture and algorithms so that knowledgeable decisions can be made when it comes time to specify the architecture of a VLSI circuit for a particular set of applications.

The systolic array testbed system is composed of a minicomputer system interfaced to the array of systolic processor elements (SPEs). The minicomputer system is an HP-1000, with the usual complement of printer, disk storage, keyboard-CRT, etc. The systolic array is housed in a cabinet approximately 28"x19"x21". The interface circuitry uses a single 16-bit data path from the host HP-1000 to communicate data and commands to the array.

Commands and data are generated in the HP-1000 by the operator using interface programs written in FORTRAN. Algorithms can be conceived, put into a series of commands for the systolic array processor, and tested for validity. Data computed in the array can be read by the host HP-1000 and displayed for the operator.

The use of a general purpose minicomputer as the driver for the systolic array gives unlimited flexibility in developing algorithms. Through the use of interface routines, algorithms can be tried,

evaluated, changed and tried again in a few minutes. Also, in cases where the output must be manipulated and fed back into the array, the manipulation of the data can be done either in the host using the high order language capability (for optimum flexibility), or in a dedicated microprocessor interfacing the systolic array to the host (for optimum speed).

INTRODUCTION

Signal processing theory currently suggests many improved processing methods which cannot be implemented in real-time because of the computational burden [1]. A partial solution is provided by faster device technology. However, parallelism will always be required when the data acquisition rate is comparable to the arithmetic cycle time and multiple operations are performed per data point. Currently trends in VLSI/VHSIC technology support the development of highly parallel computational structures to an extent which has never previously been practical. While some previously developed parallel algorithms and architectures deserve consideration because of changed economic and technical constraints, most of the algorithm and architecture development remains to be done. It is desirable to realize economies of scale for the chip set, processors, and system software. This makes it desirable to select a set of primitive operations at each level which is broad enough for wide applicability, but sufficiently structured to permit high efficiency and regularity of design with a small set of primitives. Fortunately, the computations which provide the bulk of the computational load for signal processing are highly regular in their arithmetic operations and data flow.

It has previously been shown that the major computational requirements for many important real-time signal processing tasks can be reduced to a common set of basic matrix operations [1A, 1B]. These include matrix-vector multiplication, matrix-matrix multiplication and addition, matrix inversion, solution of linear systems, least-squares approximate solution of linear systems, eigensystem solution, generalized eigensystem solution, and singular value decomposition. For implementation on a single processor, the results of extensive research in numerical linear algebra is a set of numerically stable, well documented routines for linear systems and least squares problems LINPACK [2] and one for eigensystem problems-EISPACK [3]. Parallel processor designs can utilize the available studies of the numerical stability of algorithms, but much less is known about effective parallelization of algorithms.

The array processor has been a powerful and popular augmentation of the general purpose computer, permitting the rapid implementation of operations with circulant matrices, since the eigenvectors of a circulant are the basis vectors of a discrete Fourier transform. Similarly, the eigenvectors of a Toeplitz matrix are asymptotically approximated by sinusoids, and shift-invariant linear systems have Toeplitx kernels and stationary random processes have Toeplitz covariance matrices.

Our objective is to provide a matrix processor to augment the general purpose computer and array processor, providing modular

parallelism in the hardware implementation of the equivalent of the LINPACK/EISPACK functions. This would lift the requirement of special structure in the matrices for real-time computation. On the other hand, irregular computations or data reorganization may be performed at a lower rate by the general purpose host computer, permitting a simplification of the operation set of the matrix processor.

Parallel processing architectures for the matrix operations have been surveyed [1A,1B] and it was concluded that the systolic architectures [4,5] provide the most promising combination of characteristics for utilizing VLSI/VHSIC technology for real-time signal processing: modular parallelism with throughput directly proportional to the number of cells, simple control, synchronous data flow, local interconnects, and sufficient versatility for implementing the matrix operations needed for signal processing. Previously reported systolic architectures include linear [4], hexagonal [4] and rectangular/hexagonal [5,6] arrays. The linear configurations perform matrix-vector multiplication and solution of triangular linear systems. The hexagonal configurations perform matrix multiplication/accumulation and L-U decomposition. The engagement processor rectangular/hexagonal array provides matrix multiplication/accumulation with improved efficiency for dense matrices, and may also perform as a hexagonal array for L-U decomposition [6]. Although the regularity of the arithmetic operations and data flow is high for non-partitioned operations, partitioned operations and mixed types of computations require careful consideration of "edge-effects" and incorporation of corresponding features in the hardware. An implementation using commercially available microprocessor components provides the opportunity to test a variety of algorithms on a variety of systolic array configurations, and especially to understand the architectural features needed by an algorithm's data movement and decision-making requirements. Such hardware requirements are difficult to learn by simulation alone, and expensive to learn by trial and error design of dedicated chips at the VLSI/VHSIC level of complexity.

SYSTOLIC ARRAY PROCESSOR (SAP)

The systolic array concept involves the inherent high throughput and simplicity offered by a lattice of identical processing elements, all operating in parallel on data synchronously flowing through the structure. The prospects for fabricating an array element (or several elements) on a single chip appear very good. However, many details of the algorithms, data flow, control, input/output, numerical accuracy, speed, etc. have to be determined before a particular chip design can be undertaken.

The goal of this work is to build a systolic array testbed which is flexible enough to allow experimentation with algorithms and configurations so that intelligent decisions can be made when it comes time to specify the chip architecture for a particular set of applications. The hardware implementation is shown conceptually in Figure 1. The array consists of a cabinet containing a 8-by-8 array of systolic processor circuit boards, a mother board, and a rack for the host-interface electronics.

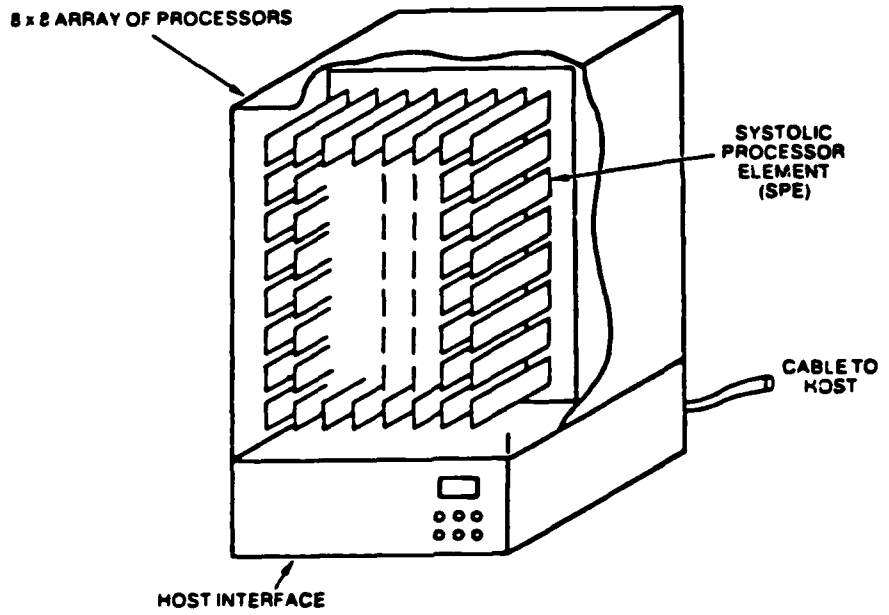


Fig. 1. Systolic array configuration.

DESIGN HISTORY AND RATIONALE

In the initial design of this testbed, many questions regarding tradeoffs and decisions had to be answered. For instance, should we use bit-serial or bit-parallel computation? what dynamic range is necessary for a useful processor? what speed of computation is reasonable? how complex or smart should each processor be? how many elements will be reasonable to connect? and what will be the packaging approach? Only the final design will be presented here, without describing the many possible alternatives.

Since the overriding concern in this testbed is for flexibility to allow experimentation, it was decided to use a microprocessor with EPROM and RAM, to allow the maximum programmability in the systolic processing element.

As for the dynamic range, consideration of signal processing uses led us to settle on a 32-bit floating-point capability. Here a tradeoff between software and hardware implementation led to the use of an Arithmetic Processing Unit (APU), e.g., Intel-8231 or AMD-9511.

Bit-serial computation was considered as a possibility because of its expandability and low pin count. However, the need for wide dynamic range and the long design time a bit-serial design would require, made the bit-parallel approach of the APU more attractive.

The type of communication path to the SPE (serial or parallel) effects the speed of operation and the hardware required for an array. Serial communication was found to be more advantageous for several reasons. Since each SPE has six I/O ports, an eight-bit parallel path would require 48 pins and 48 driver/receiver buffers in each SPE. In addition, as in VLSI design, interconnection could become a major

problem. Furthermore, to obtain flexibility in the intercommunication of the array, multiplexers have been placed in some of the data paths. The use of parallel communication would have significantly increased the amount of hardware required.

After the basic processor complexity was determined, the system was partitioned into a large (approximately 23-by-16 inch) mother board with each systolic processor on a separate 2.5-by-8 inch printed circuit board, mated to the mother board by using an edge card connector on a short side as shown in Figure 1.

SYSTOLIC ARRAY TESTBED ARCHITECTURE

The systolic array testbed system is composed of a minicomputer system interfaced to the array of systolic processor elements. The host is a minicomputer with the usual complement of printer, disk storage, keyboard-CRT, etc. The systolic array is housed in a cabinet approximately 28 by 19 by 21 inches. The interface circuitry uses a single 16-bit data path from the host minicomputer to communicate data and commands to the array.

Commands and data are generated in the host by the operator, using interface programs written in FORTRAN. Algorithms can be conceived, put into a series of commands for the systolic array processor, and tested for validity. Data computed in the array can be read by the host minicomputer and displayed for the operator.

Many other papers have discussed the theoretical aspects of the systolic array's communication of data and other properties. We have implemented the original H. T. Kung hexagonal interconnect architecture, as shown in reference [4]. By substitution of squares for hexagons, appropriate rotation of the communication paths, and realignment of the processors on a square grid, we have the square array. Now the A data paths are horizontal, the B paths are vertical, and the C paths are along a diagonal.

In this implementation, there are also virtual rows and columns along the edges of the array, as shown in Figure 2. These virtual rows and columns perform the interfacing between the parallel data path from the host and the serial communications of the systolic array processing elements. These virtual rows and columns can be thought of as existing on either side of the array, since the data paths from the SPEs on the "far side" of the array wrap around and are also connected to the left of the virtual A, B, or C rows and columns.

To achieve architectural flexibility, the serial data paths from the virtual rows and columns pass through multiplexers so that several options for data flow are possible. The data path can be selected in real time by the host processor.

It is important to note, and the essence of the great flexibility of this testbed, that thru the use of the microprocessor in the SPE we can interchange the roles of A, B and C at will.

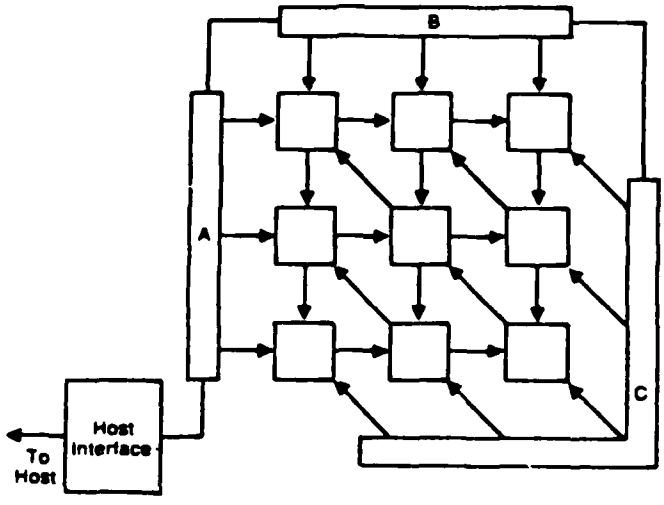


Fig. 2. The square systolic array with virtual rows and columns.

ARRAY CONFIGURATIONS

There are many array configurations available, five of which will be described here. The first is the basic engagement configuration shown in Figure 3. Here we have the data flowing in the usual A, B and C directions thru the square array of processors. This configuration will be used for (a) matrix multiplication/accumulation of full matrices [6] with no special processors and (b) L-U decomposition, which requires special boundary processors and data interchange. For instance, the top left processor performs a division while the left column outputs the C data back into the array along the A data path and the top row outputs the C data back into the array along the B data path.

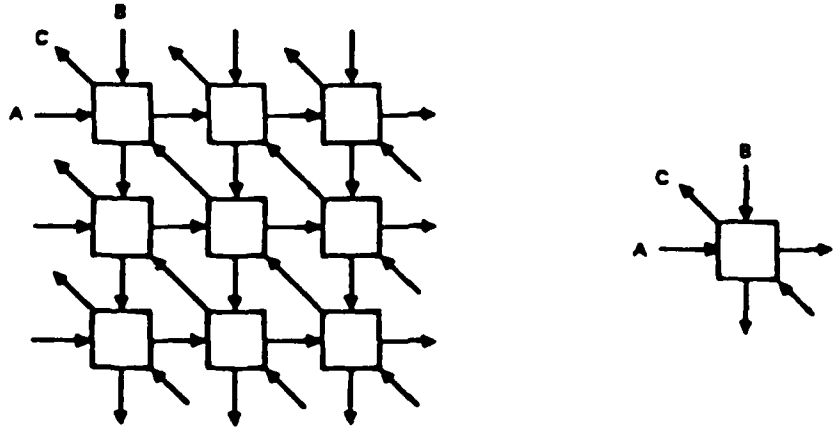


Fig. 3. Rectangular/hexagonal configuration.

The second configuration converts the square array to a linear array by routing the right output of a given row to the left input of the next row as shown in Figure 4. The routing of data is determined with multiplexers placed in the data path between the virtual rows and columns and the processors on the left and top edges. Note that in the linear array the x and y vectors move along the A data path, i.e., the rows. In performing the matrix-vector multiplication $y=Ax$ where $y=(y_i)$, $A=(a_{ij})$, and $x=(x_j)$, the matrix A is fed vertically down in an appropriate manner to form products with the elements of x which move from left to right on a given row. The product vector y moves from right to left along the row data paths which are bidirectional and time multiplexed. This configuration will be used for (a) matrix-vector multiplication with no special processors and (b) triangular linear equation solution, which requires a subtraction/division at one end of the linear array.

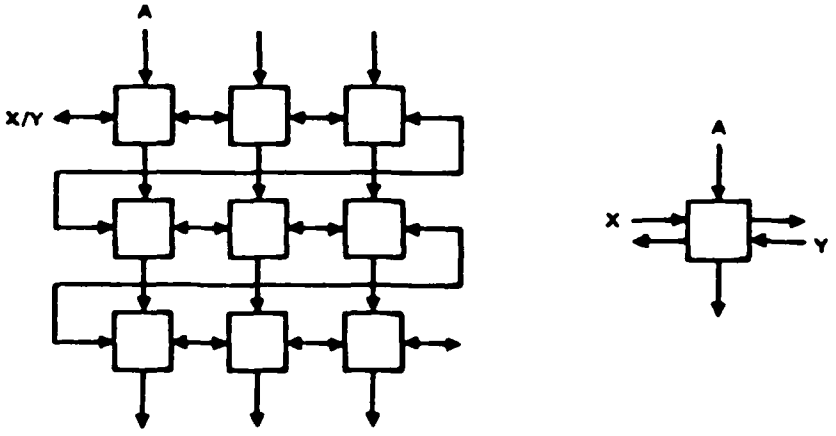


Fig. 4. Linear array configuration.

The third configuration shown in Figure 5 essentially broadcasts the same data to the top of all columns of the array. This configuration can be used to multiply a Hankel matrix by an arbitrary matrix or to form a skewed outer product [6]. A similar configuration can be used to multiply a Toeplitz matrix by an arbitrary matrix [6].

The fourth configuration, shown in Figure 6, uses a second array coupled to the first along the rows or A data path. This configuration can be used to double system throughput in the computation with complex matrices. The connection between arrays along rows enables the swapping of data from one array to the other for the computation of the AB + CD product necessary for complex matrix operations as shown in Figure 7. This configuration can be used to perform a large DFT via modular decomposition [6].

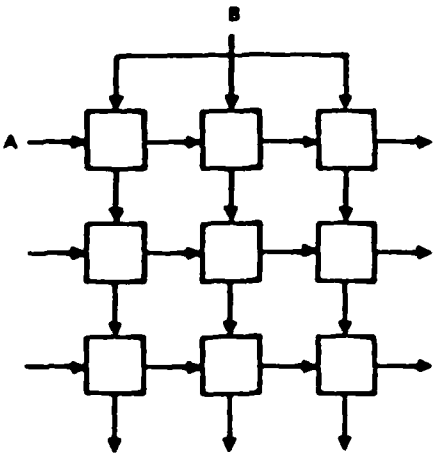


Fig. 5. Broadcast configuration.

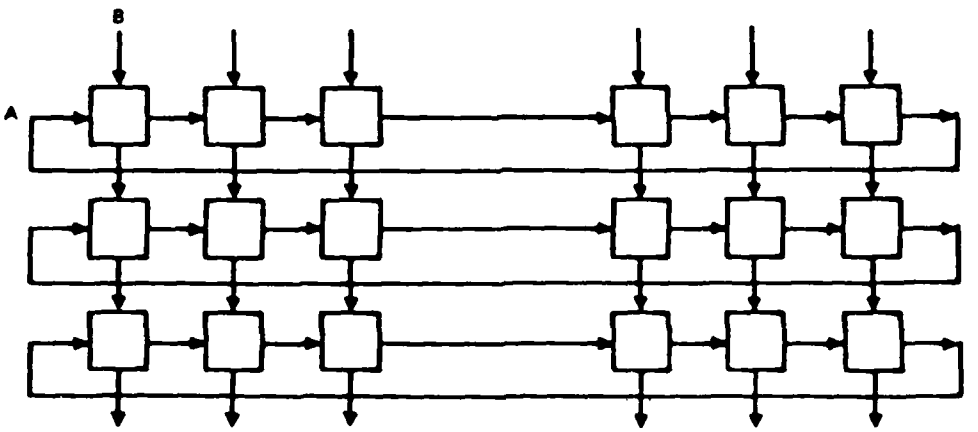


Fig. 6. Dual array configuration.

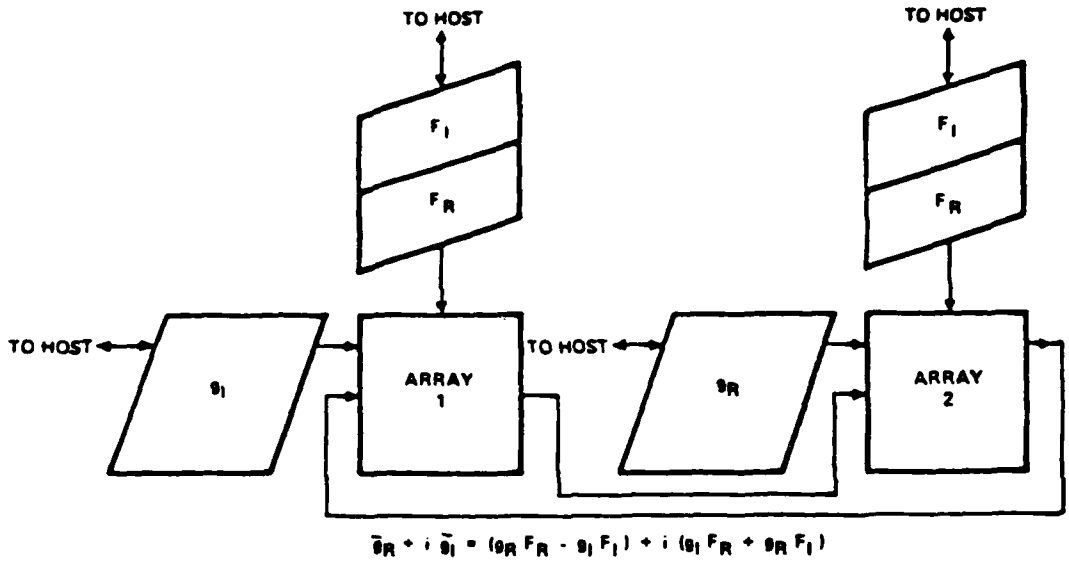


Fig. 7. Complex matrix multiplication using two real systolic arrays.

The fifth configuration is implemented in a single array and enables the transposition of a resident matrix in a straightforward manner as shown in Figure 8. Here we are utilizing the capability to interchange the roles of the A, B and C data as well as external data path multiplexers.

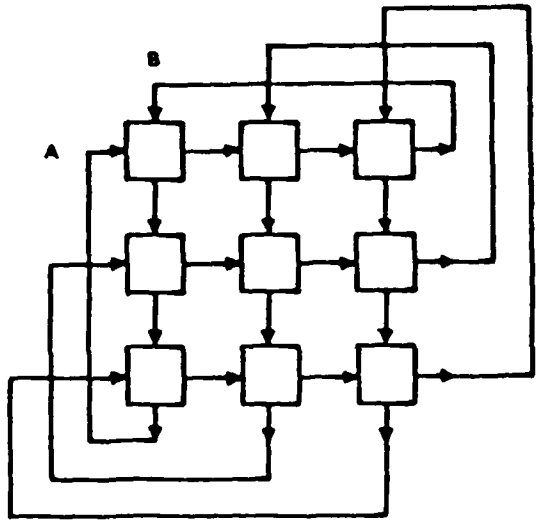


Fig. 8. Matrix transposition configuration.

THE SYSTOLIC PROCESSOR ELEMENT (SPE)

The block diagram for the systolic array processing element is shown in Figure 9. The microcomputer used is the Intel 8031. This device was chosen for its speed, internal RAM, many I/O pins, and ease of bit and byte manipulation. The data moves about in the SPE over a multiplexed 8-bit data/address bus.

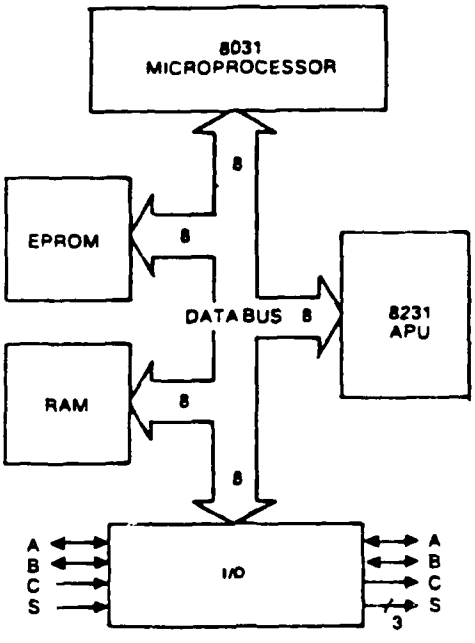


Fig. 9. Systolic processor element block diagram.

Computation is accomplished in the Arithmetic Processing Unit (APU). This unit is capable of several formats (16-bit fixed point, 32-bit fixed point, and 32-bit floating point). Many operations are available such as add, subtract, multiply, divide, square root, several trigonometric functions, etc.

The EPROM and RAM are standard devices. The EPROM is 4k X 8 bits. The RAM is 1k X 8 bits. The EPROM is used to store routines which perform data manipulation. This gives the system a hierarchial approach so that a single byte transmitted to the processor initiates a sequence of operations. The RAM can be used for data storage. This results in a "3rd dimension" of matrix storage which is important for partioned matrix operations [6]. The RAM can also be used for storage of programs during algorithm development. Once the algorithms have been perfected, they will be put into EPROM.

The I/O from the processor is bit serial. The main reason for serial I/O is to minimize pinouts and driver/receiver requirements. Four universal 8-bit parallel/serial shift registers constitute the I/O ports. Three registers are used for the A, B and C data. The fourth register, the S register, receives the instruction, broadcast along

rows, which tells the SPE which routine to perform. The I/O registers are loaded and read by the microprocessor under program control.

VLSI/VHSIC IMPLEMENTATION

It has been predicted that one or more systolic processing elements could be put on a single VLSI chip. While the currently implemented printed circuit board with 18 ICs is undoubtedly not the optimum design for the future, it is an interesting exercise to calculate the gate count for a 32-bit version of this SPE. Table 1 shows the estimated gate count for the present implementation and that which would be used in a VLSI chip if we eliminate some of the flexibility (programmability) used in this testbed. These numbers are very rough estimates, perhaps ±30%.

Table 1 - SPE Gate Count (32-bit, Floating Point)

	Testbed SPE	VLSI
Control Processor	20K	10K
APU	20K	20K
ROM	64K	16K
RAM	16K	16K
I/O	1K	2K
	121K	64K

CONCLUSION

In the last two decades numerical analysis has developed many numerically stable matrix algorithms [2,3] for use with single arithmetic unit digital computers. However, parallel numerical algorithms have not been correspondingly developed and only a limited number of parallel processors or computers have been built.

It is the belief of the authors that concurrent processing with generalized systolic architectures will provide the capability of implementing in hardware the matrix processing which currently is represented by the EISPACK [2] and LINPACK [3] software libraries. The availability of affordable VLSI matrix processing peripherals for minicomputers would significantly advance signal processing research. Similarly, VHSIC [7] implimentations of these matrix processing peripherals would make advanced signal processing available for real-time tactical signal processing applications.

Dr. Mermoz recently addressed the question of spatial signal processing beyond adaptive beamforming [8]. The conclusions of his paper were that with adequate computational resources much new information about the medium through which the signal propagates may be incorporated into the signal processing with corresponding improvements in system performance. In particular, he says, "Despite the advances in computer technology, fast as it may have been in the past decades, such a program is liable to absorb all the present capacity and probably the predictable capacity during the next twenty years. Meanwhile, there

will be some trade-offs between complexity and precision. But the trend to introducing the most flexible model, compatible with the array and the number of sources, is likely to be the right approach toward further improvements when we deal with complex and unpredictable mediums...Such an approach is rather the opposite of what has been done so far, except in advanced research. Of course, anybody would be horrified by the amount of computing power required. On the other hand, most scientists have been horrified several times in their career by what turned out to be current practice within a few years."

Systolic architectures implemented with VLSI should make this type of signal processing possible for sonar bandwidths within this decade while VHSIC implementations should provide similar capability at communication and radar bandwidths.

ACKNOWLEDGEMENTS

The authors wish to acknowledge the support of the Naval Electronics Systems Command (Elex 612, 614) and the Naval Ocean Systems Center Independent Research/Independent Exploratory Development (IR/IED) program.

REFERENCES

[1A] Speiser, J.M. and H.J. Whitehouse, "Architectures for Real-Time Matrix Operations," Proceedings of the 1980 Government Microcircuits Applications Conference held at Houston, Texas, 19-21 Nov.1980.

[1B] Speiser, J.M., H.J. Whitehouse, and K. Bromley, "Signal Processing Applications for Systolic Arrays," Record of 14th Asilomar Conference on Circuits, Systems and Computers held at Pacific Grove, California, 17-19 Nov. 1980, IEEE Catalog No. 80CH1625-3, pp 100-104.

[2] Dongarra, J.J., et al, LINPACK Users'Guide, Society for Industrial and Applied Mathematics, Philadelphia, Pennsylvania, 1979.

[3] Garbow, B.S., et al, Matrix Eigensystem Routines-EISPACK Guide Extensions, Springer-Verlag, 1977.

[4] Kung, H.T., "Systolic Arrays for VLSI," in Duff, I.S. and G.W. Stewart, Sparse Matrix Proceedings, 1978, Society for Industrial and Applied Mathematics, Philadelphia, Pennsylvania, 1979 (Reprinted in Mead, C. and L. Conway, Introduction to VLSI, Addison-Wesley, 1980).

[5] Kung, S.-Y., "VLSI Array Processor for Signal Processing," presented at Conference on Advanced Research in Integrated Circuits held at MIT, Cambridge, Massachusetts, Jan. 1980.

[6] Speiser, J.M. and H.J. Whitehouse, "Parallel processing algorithms and architectures for real-time signal processing," Real-Time Signal Processing IV, a publication of the SPIE International Technical Symposium held in San Diego, 25-28 Aug. 1981, Vol. 298.

[7] L.W. Sumney and E.D. Maynard, Jr., "The United States Department of Defense Program on Very High Speed Integrated Circuits (VHSIC)," Proc. 1979 Int. Symp. on Circuits and Systems, July 1979, pp. 559-563.

[8] H.F. Mermoz, "Spatial Processing Beyond Adaptive Beamforming," J. Acoust. Soc. Am. 70(1), July 1981, pp. 74-79.

APPENDIX D

SYSTOLIC ARRAY INTERFACE INSTRUCTION CODES

SYSTOLIC ARRAY INTERFACE INSTRUCTION CODES

MNU	DEC	HEX	OCT	DESCRIPTION
LS*	1	1	000400	Load single System register: increment to next register
LA*	2	2	001000	Load all System registers with this function code
RS	26	1A	015000	Read System registers (8 words)
BA*	28	IC	016000	Buffer Address (Operand is Bits Ø-7 of SAP buffer address)
BB*	29	1D	016400	Buffer Block (Operand is Bits 8-9 of SAP buffer address)
BE*	30	1E	017000	Buffer Enable (Bit 7: Ø=Out, 1=In; Bits Ø-6: # of bytes; Ø-1024)
RI*	31	1F	017400	Register Immediate (Operand loaded into selected register - No increment of buffer address or register)
RL	4	4	002000	Load data into register from buffer and increment
RR	5	5	002400	Read data from register into buffer and increment
MO	12	C	006000	Move Data and System registers (one clock)
MS	13	D	006400	Move System register (8 clocks)
MD	14	E	007000	Move Data registers (8 clocks)
MB	15	F	007400	Move Data and System registers (8 clocks)
SP*	6	6	003000	Select Processor multiplexer (Bit 7: Ø=Busy Enabled, 1=Disabled) (Bits Ø-1: Ø=All SPEs, 1=Columns, 2=Rows, 3=Row & Column)
SS*	7	7	003400	Select S controls (Input mux, Register Controls)
SA*	8	8	004000	Select A controls (Input mux, Register Controls)
SB*	9	9	004400	" B " " " " "
SC*	10	A	005000	" C " " " " "
SR*	11	B	005400	" Register (A, B, CA, CB: Ø to 31)
CI	24	18	014000	Clear Interface
CP	25	19	014400	Clear Processors (Reset CPU and APU)
IT	16	10	010000	Interrupt SPE
IW	17	11	010400	Interrupt SPE and Wait for Ready
WT	22	16	013000	Wait for Ready
LR*	18	12	011000	Load row register with lower byte (Bit Ø = Row 1)
LC*	19	13	011400	Load column register with lower byte (Bit Ø = Column 1)
LF*	3	3	001400	Load flag (Bit Ø: 1 = Set, 0 = Clear)
SL	20	14	012000	Slow processor clock
FS	21	15	012400	Fast processor clock
DY*	23	17	013400	Delay (16 to 256 microseconds: 1-15)
AC	27	1B	015400	Alternate Clock enables external DCLK
NO	0	0	000000	No Operation

A DATA MUX

Ø - Virtual Column (Input)
1 - Circulate Row M
2 - Row M-1
3 - Column M (Transpose)
4 - Dual Row M
5 - Dual Linear
6 - Spare
7 - Spare

B DATA MUX

Ø - Virtual Row (Input)
1 - Circulate Column N
2 - Spare
3 - Row N (Transpose)

C DATA MUX

Ø - Virtual Row (Input)
1 - Spare
2 - Spare
3 - Spare

S INPUT MUX

Ø - S Row
1 - S Column
2 - Spare
3 - Spare

REGISTER CONTROL

Ø - Hold data
1 - Shift right/down
2 - Shift left/up
3 - Load data

*Operand required in lower byte of command.

APPENDIX E

SYSTOLIC PROCESSOR ELEMENT INSTRUCTION CODES

00 NOP: SREG ONLY ENABLED
01 NOP: PREVIOUS REGISTER ENABLE
02 A REGISTER RIGHT ONLY
03 B " " "
04 A REGISTER LEFT ONLY
05 B REGISTER " "
06 C REGISTER LEFT ONLY
07 A & B REGISTER RIGHT ONLY
08 A & B REGISTER LEFT ONLY
09 A & B RIGHT & C LEFT
0A LIGHT RED LED 04C0
0B LIGHT YELLOW LED 04D2
0C LIGHT GREEN LED 04E4
*0D APU STATUS TO S REGISTER
*0E SELF TEST 0500
0F NOT USED
10 LOAD PROGRAM at 2100
11 " 2180
12 " 2200
13 " 2280
14 " 2300
15 " 2380
(Using A register input)
16 Jump to PROGRAM starting at 2100
17 " " " 2180
18 " " " 2200
19 " " " 2280
1A " " " 2300
1B " " " 2380
*1C APU to A STORE
*1D " B "
*1E " C "
*1F " W "
*20 " X "
*21 " Y "
*22 " Z "
(Inverts backward APU output)
*23 A STORE to APU
*24 B " "
*25 C " "
*26 W " "
*27 X " "
*28 Y " "
*29 Z " "
*2A CHSF CHANGE SIGN OF A
*2B FADD ADD A & B
*2C FDIV DIVIDE B BY A
*2D FLTD 32 BITINTEGER TO FF
*2E FMUL MULTIPLY A & B
*2F FSUB SUBTRACT A FROM B

*Requires Slow Processor Clock

*30 POPF STACK POP (Floating Point)
*31 PTOF " PUSH (Floating Point)
*32 PWR BA
*33 SQRT A
*34 XCHF EXCHANGE A & B
*35 CHSD SIGN CHANGE OF A
*36 DADD ADD A & B
*37 DDIV DIVIDE B BY A
*38 DMUL MULTIPLY A & B (R = LOWER 32 BITS)
*39 DMUU " " (R = UPPER 32 BITS)
*3A DSUB SUBTRACT A FROM B
*3B FIXD FLOATING POINT TO INTEGER
*3C POPD STACK POP (32-BIT INTEGER)
*3D PTOD STACK PUSH (32-BIT INTEGER)
*3E XCHD EXCHANGE A & B
3F NOT USED

40	A REG to A STORE	80	A STORE to A REG
41	B " "	81	" B "
42	C " "	82	" C "
43	S " "	83	" S "
44	A REG to B STORE	84	B STORE to A REG
45	B " "	85	" B "
46	C " "	86	" C "
47	S " "	87	" S "
48	A REG to C STORE	88	C STORE to A REG
49	B " "	89	" B "
4A	C " "	8A	" C "
4B	S " "	8B	" S "
4C	A REG to W STORE	8C	W STORE to A REG
4D	B " "	8D	" B "
4E	C " "	8E	" C "
4F	S " "	8F	" S "
50	A REG to X STORE	90	X STORE to A REG
51	B " "	91	" B "
52	C " "	92	" C "
53	S " "	93	" S "
54	A REG to Y STORE	94	Y STORE to A REG
55	B " "	95	" B "
56	C " "	96	" C "
57	S " "	97	" S "
58	A REG to Z STORE	98	Z STORE to A REG
59	B " "	99	" B "
5A	C " "	9A	" C "
5B	S " "	9B	" S "
*5C	A REG to APU	*9C	APU to A REG
*5D	B " "	*9D	" B " MS
*5E	C " "	*9E	" C " BYTE
*5F	S " "	*9F	" S " 1ST

60 THRU 65 NOT USED
A0 THRU FF NOT USED
*66 DMAC - DBL. PRECISION MULTIPLY & ACCUMULATE
*67 EPDM - ENGAGEMENT PROCESSOR DATA MOVE
68 THRU 7F NOT USED

APPENDIX F

HIGH LEVEL S-CODES FOR THE SPE - PRELIMINARY

The Interrupt Service routine which sets the busy line will turn on the green LED (Bit-Ø of the C-Port) whenever the busy line is set. The green LED will be turned off when the command has been completed. Data movement thru the C-Port will be performed, if required, without turning on the LED again. The C-Port will be left in a Hold-Data mode to prevent shifting.

DMAC-66 - Double Precision Multiply and Accumulate:
This code performs the A times B plus C function, assuming that the data is in 32-bit integer format.

First the A-Store and B-Store are put into the APU. They are multiplied together and then the C-Store data is put into the APU, and added to the A times B product. The result is then stored in the C-Store. A- and B-Stores are not changed.

FMAC-65 - Floating Point Multiply and Accumulate:
This is identical to DMAC[66] except that the data is treated as floating point.

EPDM-67 - Engagement Processor Data Move:
This code moves the A-Store data along the rows and the B-Store data along columns.

Several results can be achieved depending on the shift directions set into the ports and the virtual row and column registers in the interface. Also, selected rows and columns can be moved by only enabling certain SPEs.

For instance, EPDM can be used for the matrix-matrix multiplication when the matrices are coming in from the virtual column and virtual row or when the data is already within the array and only being rotated.

The operation of the command is as follows: First the shift directions are set up. Then the EPDM [67] S-Code is sent to the

desired SPES. An Interrupt causes the SPE to load the A- and B-Ports with the first byte of the A- and B-Store data. Then a 'Move-Flag-Wait' cycle moves the byte and causes the SPE to read the byte, store it in the A- and B-Stores and read the next bytes and put them in the respective ports for the next 'Move-Flag-Wait' cycle. This is repeated four times to move the 32-bit data word.

ALAT-74 - A-Store Lateral Move:

This command will move the A-Store data laterally one column. It is identical to EPDM except that only the A-Store data is moved.

APUSA-ØF - APU Status to A-Port:

Move the APU Status byte to the A-Port. S-Code 'OD' moves the Status byte to the S-Port. The A-Port is more convenient in some cases.

IRMRD-64 - Internal RAM Read:

A three byte command. The first byte is the S Code '64' which is put into the S-Port. Then an interrupt is sent.

The second byte is the internal RAM starting address which is placed in the S-Port and followed with a Flag cycle.

The third byte is the internal RAM ending address which is placed in the S-Port and followed with a Flag cycle.

The first data byte is now in the A- and B-Ports ready to be moved into the interface register for storing into the SAP buffer. When all the A- or B-Ports have been stored in the SAP buffer, another 'Flag-Wait' cycle gets the next byte.

The SPE will count the bytes output. When the end address has been reached the SPE will return to the idle state with the previous shift condition restored. The SPE will ignore any more Flags.

XSCDS-FE - Execute S-Codes:

This command causes the CPU to go to the external RAM address starting at 2ØØH and interpret the contents as a string of S-Codes which are executed one at a time just as if they came from the S-Port.

All the Flags and data movements required for a Fetched S-Code are the same. The process continues until a 'FF' is fetched which terminates the command.

ALCLR-AC - All Clear:

This command loads zeros into the internal CPU RAM and into the 1Ø24 bytes of external RAM.

LDRAMØ-68 -Load External RAM:

" 1-69
" 2-6A
" 3-6B
" 4-6C
" 5-6D
" 6-DE
" 7-DF

This command loads data into the specified block [Ø-7]. Data will be loaded into non-CPU RAM from the S-Port starting at the block address specified by the command.

There are eight blocks in the external RAM. A block is 128 bytes. This command will continue to input bytes until the processor has input 128 bytes or the SPE is reset. The Flag line will be used as the Transfer Acknowledge Signal.

The S-Port is loaded with the appropriate byte and a 'Flag-Wait' cycle performed. The SPE will take the S-Port data and store it into external RAM. This cycle can be repeated until 128 bytes have been input or until a reset signal is sent to the SPE.

RDRAMØ-78 -Read External RAM:

" 1-79
" 2-7A
" 3-7B
" 4-7C
" 5-7D
" 6-7E
" 7-7F

Read RAM. This command is similar to the LDRAM command except that with each Flag line cycle, the next RAM byte is read and put into the A- and B-Ports in order to be read out to the SAP interface buffer.

TSTOAS-3F - Test Top-Of-APU-Stack:

This command tests the data on the top of the APU stack. If the data is positive, the red LED will be turned on. If the data is negative, the yellow LED will be turned on. If the data is zero, the red and the yellow LED will be shut off.

DISABL-FD - DISABLE SPE:

This command disables the processor by putting it into a loop in which the processor ignores all Interrupts and Flags. The reset signal will get the SPE back into the Ready state.

NB: The following codes are specifically designed for use during the Singular Value Decomposition algorithm.

SVD1-A0 - SVD START:

This command initializes the SVD algorithm. The norms and their inner products are calculated. The decision is made whether or not to do a rotation. If a rotation is required, the sine and cosine are calculated and both the data and the identity matrix columns resident in the SPE are processed.

The function ends with the B-Port in a shift left (up) condition, containing a status byte with the following meaning:

ØØH - No rotation and no error
Ø1H - Rotation performed
Ø8H - Error: Square root of a negative number
Ø4H - ØCH - 14H - 1CH - Error: Underflow
Ø2H - ØAH - 12H - 1AH - Error: Overflow

If an error occurred during the operation, besides showing up in the B-Port, the green LED will stay on until an operation is successfully completed. The point in the program where the error occurred can be determined by checking the internal CPU RAM location 3CH and 3DH.

SVD Step Two:

This command is essentially the same as the SVD1 command except that some initialization is not done. The test for rotations is performed and rotations done if necessary.

This command is used to complete a sweep, i.e., cycle thru all combinations of columns. If no rotations are performed, the B-Port always contains ØØH. This means the algorithm has converged and further processing is useless.

Thus to perform a sweep on an 8-by-8 matrix, SVD1 is used once and SVD2 is used six times with column swaps in between. Refer to the LSDD thru RSDU commands for swapping the columns of the matrices.

SVD3-A2 - SVD Final Step:

This command is used when the array has been orthogonalized as evidenced earlier by a complete sweep with no rotations.

In this function the diagonal elements (ZL and ZR) are computed and both the U and the V matrix columns are normalized if the respective Z is greater than the tolerance. If normalization is not desired or causes a problem with overflow, then a large value of the tolerance can be input using the TOLIN [A7] command. This command also uses the current value of depth which can be changed using the depth [A6] command.

As in SVD1 and SVD2 the B-Port is left with ØØH or an Error code if something goes wrong and the green LED is left in the ON state.

DEPTH-A6 - Depth of Column:

This command sets the number of rows present in the array to be processed with the SVD algorithm. The default value is eight.

The command is executed as follows: The S-Code 'A6' is sent to the appropriate SPEs followed with an Interrupt. The Number of Rows is then placed in the SPE's S-Port. The value is in hexadecimal format. The Flag line is then cycled down and up and a Wait used to allow the SPE to complete the command.

TOLIN-A7 - TOLERANCE INPUT:

This command loads the tolerance value for the SVD algorithm.

The command is executed as follows: The S-code 'A7' is sent to the appropriate SPEs followed by an Interrupt. Then the four bytes of

the 32-bit floating point tolerance value are input into the S-Ports, each followed by a 'Flag-Wait' cycle.

The Default value 57 8Ø ØØ ØØH = 2**-41*Ø.5 = 2.27 E-13 is set into the CPU internal RAM when the SVD1 [AØ] command is executed.

PKLD-BØ - Pack Data/Unit Matrix:
PKRD-B2
PKLU-B4
PKRU-B6

This command packs the A-Store into one of four areas in the SPE's external RAM for the SVD process. See the description of the SVD for details of the process.

These commands do two things. First, the value in the executing SPE's own A-Store is moved to the appropriate location in RAM. Second, the data from the seven SPE's below the executing SPE are moved up into the top SPE through the B-Port and stored in its external RAM.

The data locations in RAM are as follows:

Left Data - 21ØØH to 213FH
Right Data - 214ØH to 217FH
Left Unit - 218ØH to 21BFH
Right Unit - 21CØH to 21FFH

Note that space has been assigned so that a 16-by-16 array could be processed.

XPLD-B1 - Expand Data/Unit Matrix:
XPRD-B3
XPLU-B5
XPRU-B7

These commands expand the data packed into the top Row of SPEs.

Two operations are performed by each of these commands. First, the data corresponding to the top element in the column is stored in the A-Store of the SPE. Second, the 28 bytes of the seven 32-bit (4 byte) data is placed into the B-Port for moving down the column to the appropriate SPE.

LSDD-A3 - Left/Inside/Right Side Data/Unit Column Swap:
ISDD-A4
RSDD-A5
LSDU-A8
ISDU-AO
RSDU-AA

These codes are used to swap the data and unit columns in the SVD algorithm.

The Left Side code is sent to the left-most SPE. The Inside code is sent to the inner SPEs. The Right Side code is sent to the right-most

SPE. An 'Interrupt-and-Wait' [IW] instruction is sent to the array. Each SPE will put a byte into the A-Port which should be shifted to the Right. A 'Move-Flag-Wait' cycle is sent to the SAP. After the move, the Flag will cause the SPEs to read the A-Port and store the byte in the appropriate position for data coming in from the left. It will then put a byte of data which should be moved to the left into the A-Port. The next 'Move-Flag-Wait' cycle will move that byte to the left where the SPE will read it and store it in the appropriate location.

This is repeated until both a column swap to the right and a column swap to the left have been completed. In the 8-by-8 array, a column of data is 8-by-4 bytes long so 32 'Move-Flag-Wait' cycles are required for both the data swap and the unit matrix swap.

RDLD-60 - Read Left [Right] Data [Unit] Columns:
RDRD-61 These commands read the left or right data columns out thru the A- or
RDLU-62 B-Port.
RDRU-63

The appropriate S-Code is placed into the S-Port of the desired SPEs and an 'Interrupt and Wait' is sent to the SAP. The first data byte of data will be put into its A- and B-Ports by executing SPE.

The data bytes should then be read out of the A- or B-Port and stored in the appropriate locations in the SAP buffer.

The rising edge of the Flag line will cause the next byte to be read from the RAM. This process repeats until the whole column, 32 bytes, has been read out.

LDLD-70 - Load Left [Right] Data [Unit] Columns:
LDRD-71 These commands load the left or right data columns thru the B-Port
LDLU-72 from the SAP buffer.
LDRU-73

After the S-Port is loaded with the appropriate code an Interrupt and Wait is sent to the SAP. The first byte of data is read from the SAP

buffer and put into the B-Ports of the SPEs and a 'Flag-Wait' cycle is performed. The SPE will read the B-Port and store the byte in the appropriate cell in its external RAM.

This process repeats until the whole column of 32 bytes has been loaded into the RAM.